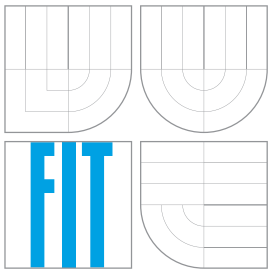
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

IMPLEMENTACE STATISTICKÝCH KOMPRESNÍCH METOD

IMPLEMENTATION OF STATISTICAL COMPRESSION METHODS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JIŘÍ ŠTYS

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. DAVID BAŘINA

BRNO 2013

Abstrakt

Tato diplomová práce popisuje Burrowsův-Wheelerův kompresní algoritmus. Detailně se zaměřuje na jednotlivé části Burrowsova-Wheelerova algoritmu, nejvíce na transformaci globální struktury a entropické kódery. V rámci transormace globální struktury jsou popsány například tyto metody presuň na začátek, inverzní frekvence, intervalové kódování a další. Mezi popsanými entropickými kodéry jsou Huffmanovo, aritmetické a Riceovo-Golombovo kódování. V závěru je provedeno testování metod transformace globální struktury a entropických kodérů. Nejlepší kombinace je porovnána s nejpoužívanějšími kompresními algoritmy.

Abstract

This thesis describes Burrow-Wheeler compression algorithm. It focuses on each part of Burrow-Wheeler algorithm, most of all on and entropic coders. In section are described methods like move to front, inverse frequences, interval coding, etc. Among the described entropy coders are Huffman, arithmetic and Rice-Golomg coders. In conclusion there is testing of described methods of global structure transformation and entropic coders. Best combinations are compared with the most common compress algorithm.

Klíčová slova

Komprese dat, kódování delkou sledů, Burrowsova-Wheelerova transformace, inverzní frekvence, přesuň na začátek, Huffmanovo kódování, aritmetické kódování, Rice-Golombovo kódování, intervalové kódování, vážený frekvenční počet, distatnční kódování, statické kompresní metody, Lubyho transformace.

Keywords

Data kompression, run-length encoding, Burrows-Wheeler transform, inversion frequencies, move to front, Huffman coder, arithmetic coder, Rice-Golomb coder, interval endoding, weighted frequency count, distance coding, statistic compression methods, Luby transform.

Citace

Jiří Štys: Implementace statistických kompresních metod, diplomová práce, Brno, FIT VUT v Brně, 2013

Implementace statistických kompresních metod

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Davida Bařiny.

. . . . . .
Jiří Štys
22. května 2013

Poděkování

Rád bych poděkoval vedoucímu práce panu Ing. Davidu Bařinovi za pomoc s diplomovou prací.

Obsah

Kapitola 1

Úvod

Cílem této diplomové práce je seznámení s kompresními metodami. Podrobně je zde popsán Burrowsův-Whellerův kompresní algoritmus s jeho modifikacemi.

V první kapitole jsou popsáno co je vlastně komprese dat. Je zde zmíněno rozdělení komprese dat na bezeztrátovou a ztrátovou kompresi a nejznámější představitelé pro jednotlivé druhy kompresí.

Druhá kapitola se věnuje samotnému Burrowsově-Wheelerově kompresnímu algoritmu. V této kapitole je popsána Burrowsova-Wheelerova transformace. Jsou zde popsány různé metody transformace globální struktury například přesuň na začátek, distanční kódování a další. Následuje vysvětlení kódování délkou sledů. Popis Burrowsova-Wheelerova kompresního algoritmu zakončuje popis některých entropických kodéru aritmetického a Huffmanova kódování. V závěru kapitoly jsou nastíněny možne modifikace metod transformace globální struktury.

Ve třetí kapitole jsou testovány implementované metody transformace globální struktury na korpusech Calgary, Caterbury a Silecia. Jednotlivé testy jsou srovnávány na základě kompresního poměru a časové náročnosti. Další testovaní se věnuje rozdílům mezi jednotlivými etropickými kodéry. Je zde i ověřován vliv velikosti bloku řazených dat v Burrowsově-Wheelerově transformaci. Jsou zde i testy k modifikované varianě Burrowsovu-Wheelerovu kompresnímu algoritmu, kde se zaměňuje pořadí metod. Na závěr této kapitoly jsou porovnány nejúčinnější kompresní metody s aktuálně používanými kompresními algoritmy.

Kapitola 2

Komprese dat

Se zvyšujícími objemy dat roste potřeba úspory prostoru na mediích, na kterých mají být data uložena. Z toho důvodu se začaly vyvíjet metody, které by zmenšili objem ukladaných dat. Kompresní algoritmy se snaží odstranit nadbytečné informace z dat. Tyto algoritmy jsou popsány v sekci 2.2. Komprese dat se využívá jak pro ukládání dat na medium, tak i pro přenos přes síť s omezenou rychlostí.

2.1 Základní pojmy

V tomto odstavci je vysvětleno několik pojmů, které jsou použity v této práci. Pod pojmem kód a kódování rozumíme způsob reprezentace dat, informací a hodnot, která jsou uložena v paměti nebo souboru atd. Znak je základní jednotkou dat. Znakem se v teorii komprese se nazývá libovolná hodnota uložená v jednom bytu. Textem a vstupním řetězcem jsou označována data, která mají být komprimována. Nemusí jít o textovou informaci. Kompresní poměr slouží k porovnání výkonnosti kompresních metod. Tento poměr je popsán rovnicí 2.1.

$$\text{kompresní poměr} = \frac{\text{velikost výstupního toku}}{\text{velikost vstupního toku}} \tag{2.1}$$

Hodnota vypočítaná vzorcem 2.1 nám udává, jestli došlo ke kompresi či k expanzi. Pokud bychom získali hodnotu 0,5, tak výstupní komprimovaný text bude mít 50% vstupního textu. Při hodnotách pod 1 dochází ke kompresi. Nad hodnotu 1 dojde k expanzi textu. Hodnoty, které budou uvedeny ve výsledcích jednotlivých modifikací, budou uvedeny v *bpc (bits per character)*. Tato veličina udává průměrný počet výstupních bitů potřebných k reprezentaci jednoho znaku viz. rovnice .

$$\text{kompresní poměr} = \frac{\text{velikost výstupního toku} \cdot 8}{\text{velikost vstupního toku}} \tag{2.2}$$

2.2 Typy kompresních metod

Kompresní metody se dělí podle jejich přístupu k datům a jejich zpětné obnově. Podle tohoto přistupu se dělí na dvě základní skupiny.

2.2.1 Bezeztrátová komprese

Hlavní charakteristickou vlastností těchto metod je, že při kompresi textu a opětovné rekonstrukci dostaneme totožná data, jako byla na vstupu. Tyto metody se používají kdekoli, kde by jakákoli ztráta kvality vedla k nenávratnému poškození textu. Formáty ZIP, RAR, gzip, bzip2 atd. využívají tuto metodu. Bezeztrátové metody se dále dělí podle způsobu komprese na statické a slovníkové.

Statické metody

Statické metody jsou založeny na pravděpodobnosti výskytu jednotlivých znaků, kterým přidělují kód s proměnnou délkou. Znaky s vyšší pravděpodobností mají kratší kód oproti těm s nižší pravděpodobností. Pravděpodobnost znaků je buď známa před kompresí nebo se používá prediktor, který podle předcházejících znaků predikuje následující. Typickými představiteli jsou Huffmanovo a Aritmetické kódování.

Slovníkové metody

Slovníkové metody vyhledávají opakující se části textu. Při prvním výskytu se zapíše celá sekvence do zkomprimovaného textu. U následujících výskytů je tato sekvence nahrazena odkazem na předchozí výskyt. Představiteli jsou LZ77, LZX, LZW.

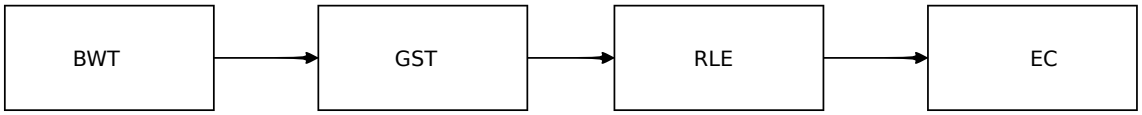
2.2.2 Ztrátová komprese

Tyto metody dosahují vyššího kompresního poměru na úkor ztráty některých méně významných informací. Ztrátová komprese je vhodná ke kompresi obrázku, videa a zvuku. U těchto typů dat lze lehce potlačit některá méně významná data pro člověka, například u zvuku frekvence, které člověk nemůže vnímat.

Kapitola 3

Burrowsův-Wheelerův kompresní algoritmus

Burrowsův-Wheelerův kompresní algoritmus se skládá ze čtyř částí, které jsou vidět na obrázku 3.1. Každá z těchto částí je reverzibilní [5]. Existuje mnoho variant, některé zde budou popsány.



Obrázek 3.1: Složení Burrowsova-Wheelerova kompresního algoritmu

Při kompresi se postupuje z levé strany do prava. Jednotlivá výstupní data jsou zároveň vstupní do dalšího bloku. První blok tvoří Burrowsova-Wheelerova transformace, která má za úkol seřadit data a stejné symboly dát do posloupností, čímž vzniknou lokální shluky stejných znaků. Na převod z lokálních kontextů na globální slouží druhý blok Burrowsova-Wheelerova algoritmu, který se nazývá transformace globální struktury. Poté následuje kódování délkou sledů, které nahrazuje shluky stejných znaků daným způsobem nejčastěji dvojicí (znak, počet opakování). Posledním blokem je entropické kódování, které komprimuje jednotlivé symboly. Jako entropický kodér se používá aritmetické nebo Huffmanovo kódování.

3.1 Burrowsova-Wheelerova transformace

Burrowsova-Wheelerova transformace tvoří první blok celého algrotimu. V roce 1994 tuto transformaci vyvinuli Michael Burrows a David Wheeler. Burrowsova-Wheelerova transformace zpracovává vstupní text po blocích a je reverzibilní. Každý blok má předem definovanou maximální velikost a je zpracováván odděleně od ostatních bloků. Tato metoda je vhodná pro soubory větší jak 1000 znaků. Hlavní myšlenkou Burrowsovy-Wheelorovy metody je transformace vstupního textu na nový text, který obsahuje shluky stejných znaků. Díky těmto shlukům se zlepšuje možnost koprese. Transformace je provedena pomocí permutací vstupního textu a jejich vhodného seřazení takového, aby bylo možné znovu sestavit původní řetězec.

Uvedeme si příklad práce Burrowsovy-Wheelerovy transformace. Na vstupu máme text *mississippi*. Podle délky vstupního textu N si vytvoříme matici o rozměrech NxN. Do jejího prvního řádku vložíme vstupní řetězec. Každý další řádek bude obsahovat hodnoty předcházejícího řádku orotovaného o jeden znak doleva.

$$A_x = \begin{pmatrix} m & i & s & s & i & s & s & i & p & p & i \\ i & s & s & i & s & s & i & p & p & i & m \\ s & s & i & s & s & i & p & p & i & m & i \\ s & i & s & s & i & p & p & i & m & i & s \\ i & s & s & i & p & p & i & m & i & s & s \\ s & s & i & p & p & i & m & i & s & s & i \\ s & i & p & p & i & m & i & s & s & i & s \\ i & p & p & i & m & i & s & s & i & s & s \\ p & p & i & m & i & s & s & i & s & s & i \\ p & i & m & i & s & s & i & s & s & i & p \\ i & m & i & s & s & i & s & s & i & p & p \end{pmatrix}$$

Seřadíme jednotlivé řádky matice A_x v abecedním pořadí. Získáme novou matici A_{x1}. Musíme si zapamatovat řádek, na kterém je uložen text v původní podobě, jinak bychom nemohli zakódovaný text dekódovat.

$$A_{x1} = \begin{pmatrix} i & m & i & s & s & i & s & s & i & p & p \\ i & p & p & i & m & i & s & s & i & s & s \\ i & s & s & i & p & p & i & m & i & s & s \\ i & s & s & i & s & s & i & p & p & i & m \\ m & i & s & s & i & s & s & i & p & p & i \\ p & i & m & i & s & s & i & s & s & i & p \\ p & p & i & m & i & s & s & i & s & s & i \\ s & i & p & p & i & m & i & s & s & i & s \\ s & i & s & s & i & p & p & i & m & i & s \\ s & s & i & p & p & i & m & i & s & s & i \\ s & s & i & s & s & i & p & p & i & m & i \end{pmatrix}$$

Zakódovaný text, který bude výstupem Burrowsovy-Wheelerovy transformace je v posledním sloupci. Zároveň musíme dát na výstup index řádku, na kterém je vstupní text v původním znění. Výstupní text by mohl vypadat například takto *5pssmipissii*.

3.1.1 Inverzní funkce

Popíši zde dvě možnosti, jak dekódovat soubor kódovaný Burrowsovou-Wheelerovou transformací.

První metoda

Začneme první a jednodušší metodou. Nejdříve si popíšeme inverzní funkci, a pak si jí ukážeme na příkladu. Představme si, že jednotlivé znaky budeme načítat do matice tzn. jeden znak na řádek. V prvním kroku si načteme vstupní text, který máme dekódovat. Seřadíme ho podle abecedy. Ve druhém kroku posuneme načtené znaky o sloupec doprava a do vzniklého sloupce načteme opět vstupní text. Jednotlivé řádky seřadíme dle abecedy. Druhý krok opakujeme do té doby, dokud nám nevznikne matice NxN. V posledním kroku

načteme znak, který se nacházel na poslední pozici. Řádek, který obsahuje tento znak v posledním sloupci, je náš zakódovaný text. Tato metoda je vhodná pro případy, kdy si můžeme zvolit speciální znak, který představuje konec kódovaného textu.

Nyní si tento postup předvedeme na příkladu. Postup si ukážeme na textu *mississippi*, ke kterému přidáme na konec speciální znak *mississippi@*. Po zakódování bude tento text vypadat následovně *ipssm@pissii*. Zároveň tento text je i vstupním textem. Načteme tento text do matice.

iterace	1		2		3		4	
operace	přidání	seřazení	přidání	seřazení	přidání	seřazení	přidání	seřazení
	i	@	i@	@m	i@m	@mi	i@mi	@mis
	p	i	pi	i@	pi@	i@m	pi@m	i@mi
	s	i	si	ip	sip	ipp	sipp	ippi
	s	i	si	is	sis	iss	siss	issi
	m	i	mi	is	mis	iss	miss	issi
	@	m	@m	mi	@mi	mis	@mis	miss
	p	p	pp	pi	ppi	pi@	ppi@	pi@m
	i	p	ip	pp	ipp	ppi	ippi	ppi@
	s	s	ss	si	ssi	sip	ssip	sipp
	s	s	ss	si	ssi	sis	ssis	siss
	i	s	is	ss	iss	ssi	issi	ssip
	i	s	is	ss	iss	ssi	issi	ssis

Tabulka 3.1: Dekódování Burrowsovy-Wheelerovy transformace

V poslední iteraci bude matice vypadat následovně:

$$\begin{pmatrix} i & @ & m & i & s & s & i & s & s & i & p & p \\ p & i & @ & m & i & s & s & i & s & s & i & p \\ s & i & p & p & i & @ & m & i & s & s & i & s \\ s & i & s & s & i & p & p & i & @ & m & i & s \\ m & i & s & s & i & s & s & i & p & p & i & @ \\ @ & m & i & s & s & i & s & s & i & p & p & i \\ p & p & i & @ & m & i & s & s & i & s & s & i \\ i & p & p & i & @ & m & i & s & s & i & s & s \\ s & s & i & p & p & i & @ & m & i & s & s & i \\ s & s & i & s & s & i & p & p & i & @ & m & i \\ i & s & s & i & p & p & i & @ & m & i & s & s \\ i & s & s & i & s & s & i & p & p & i & @ & m \end{pmatrix}$$

Nyní načteme znak, který označuje konec *@*. Vyhledáme příslušný řádek, na kerém je v posledním sloupci koncový symbol. Hledaný řádek je na paté pozici. Text, který byl původně kódován, je *mississippi@*. Dekódování bylo úspěšné, oba texty se shodují.

Druhá metoda

Popíšeme si druhou možnost, jak dekódovat text, jenž byl zakódován Burrowsovou-Wheelerovou transformací. V prvním kroku načteme vstupní soubor. U každého z načtených znaků (sloupec *F*) si zapamatujeme jeho pozici (sloupec *pozice*) ve vstupním souboru. Poté tyto dvojice

seřadíme podle abecedy (sloupec L), v případě shody porovnáme podle pozice ve vstupním textu viz. tabulka 3.2. Sloupec T obsahuje pozici ve vstupním souboru. Druhý parametr při řazení je důležitý. Bez něho mohou vzniknout v sestavované tabulce lokální smyčky (dva znaky, které ukazují jeden na druhého). A při dekódování pomocí vztahu 3.1 zamezíme výskytu těchto smyček a tím úspěšně dekódujem text.

pozice	F	T	L
0	i	5	@
1	p	0	i
2	s	7	i
3	s	10	i
4	m	11	i
5	@	4	m
6	p	1	p
7	i	6	p
8	s	2	s
9	s	3	s
10	i	8	s
11	i	9	s

Tabulka 3.2: Dekódování Burrowsovy-Wheelerovy transformace - 1. krok

Po zkonstruování tabulky 3.2 budeme pokračovat druhým krokem, ale nejdříve si musíme definovat vztah, jímž budeme procházet vytvořenou tabulku.

$$S[i] \leftarrow L[T^i[I]] | i = 0, 1, \cdots, n-1; T^0[j] = j; T^{i+1}[j] = T[T^i[j]] \tag{3.1}$$

Proměnná n představuje délku vstupního textu, S výstupní text, i krok iterace, L, T sloupce v tabulce 3.1.

Druhý krok je pouze využití definovaného vztahu 3.1. Hodnota proměnné j na začátku je nastavena na pozici posledního prvku v nekódovaném textu, proto si tuto pozici musíme pamatovat. Aplikace vztahu bude vypadat následovně. V našem případě bude hodnota proměnné $j = 5$.

1. $S[0] = L[T^0[I]] = L[T^0[5]] = L[5] = m$
2. $S[1] = L[T^1[I]] = L[T[T^0[I]]] = L[T[[5]]] = L[4] = i$
3. $S[2] = L[T^2[I]] = L[T[T^1[I]]] = L[T[[4]]] = L[11] = s$
4. $S[3] = L[T^3[I]] = L[T[T^2[I]]] = L[T[[11]]] = L[9] = s$
5. $S[4] = L[T^4[I]] = L[T[T^3[I]]] = L[T[[9]]] = L[3] = i$
6. $S[5] = L[T^5[I]] = L[T[T^4[I]]] = L[T[[3]]] = L[10] = s$
7. $S[6] = L[T^6[I]] = L[T[T^5[I]]] = L[T[[10]]] = L[8] = s$
8. $S[7] = L[T^7[I]] = L[T[T^3[I]]] = L[T[[8]]] = L[2] = i$
9. $S[8] = L[T^8[I]] = L[T[T^7[I]]] = L[T[[2]]] = L[7] = p$

10. $S[9] = L[T^9[I]] = L[T[T^8[I]]] = L[T[[7]]] = L[6] = p$
11. $S[10] = L[T^{10}[I]] = L[T[T^9[I]]] = L[T[[6]]] = L[1] = i$
12. $S[11] = L[T^{11}[I]] = L[T[T^{10}[I]]] = L[T[[1]]] = L[0] = @$

Po dokončení všech iterací získáme původní text, který byl kódován v našem případě je to text *mississipi@*.

3.1.2 Implementace - Kódování

Implementaci kódování pomocí Burrowsovy-Wheelerovy metody si ukážeme na pseudokódu a pak si ho i popíšeme.

```
while(!KonecSouboru)
{
  ulozZnak(nactiZnakZeSouboru(VstupníSoubor));
  if (PlnyBlok)
  {
    seradBlok();
    vypisPosledniZnaky(VystupniSoubor);
  }
}
seradBlok();
vypisPosledniZnaky(VystupniSoubor);
```

Pseudokód 3.1: Kódování pomocí Burrowsovy-Wheelerovy transformace

Funkce *ulozZnak()* nejenže ukládá načtený znak (který je předán jako parametr) do bloku, ale i ukazatel na tento znak. Pole ukazatelů do bloku nám bude sloužit pro řazení a jednoduché vytvoření permutací daného bloku. Zapamatovává si pozici znaku, který byl buď posledním v souboru nebo poslední před naplněním bloku. Funkce *seradBlok()* bude řadit pole ukazatelů. Záleží na implementovaném řadícím algoritmu, ale postup při porovnání dvou ukazatelů je následující. Každý z ukazatelů je zároveň i ukazatelem na první znak orotovaného bloku. Nám tedy stačí vytvořit řetězec od tohoto znaku a porovnat ho s druhým, takto vytvořeným. Nebo rychlejší metodou, postupně začít porovnávat znaky od těchto ukazatelů a posouvat se vždy při shodě o jeden znak. V případě neshody porovnat tyto dva znaky, které nám dají výsledek porovnání těchto dvou ukazatelů. Zároveň nám říká, zda máme tyto dva ukazatele mezi sebou zaměnit. Funkce *vypisPosledníZnaky()* zapíše pozici posledního načteného znaku a poslední znaky orotovaného bloku do vystupního souboru, který je předán jako parametr. Které získá tak, že projde seřazené pole ukazatelů a vždy jen vypsíše předcházející znak.

3.1.3 Implementace - Dekódování

Dekódování Burrowsovy-Wheelerovy transformace by se dalo popsat následovně pomocí pseudokódu.

```
while(!KonecSouboru)
{
   ulozZnak(nactiZnak(VstupniSoubor));
   if(PlnyBlok)
   {
        serad();
        zrekonstruujBlok();
        vypisBlok(VystupniSoubor);
   }
}
serad();
zrekonstruujBlok();
vypisBlok(VystupniSoubor);
}
```

Pseudokód 3.2: Dekódování pomocí Burrowsovy-Wheelerovy transformace

Funkce *nactiZnak()* v prvním kroku načte pozici posledního znaku a pak již jen načítá znaky a ukládá si je společně s jejich pozicemi. Načítání probíhá ze vstupního souboru, který je předán funkci jako parametr. Funkce *ulozZnak* ukládá znaky, které získá od fuknce *nactiZnak*. Načtený blok se abecedně seřadí pomocí funkce *serad*, ale nesmí se zapomenout, že při shodě znaků rozhoduje jejich pozice ve vstupním souboru. Poté je provedeno zrekonstruování načteného bloku funkcí *zrekonstruujBlok*. Následuje již jen vypsání zrekonstruovaného textu do výstupního souboru funkcí *vypisBlok*. Případně se tento celý proces opakuje, dokud nejsou načteny všechny znaky ze vstupního souboru.

3.2 Transformace globální struktury

Transformace globální struktury je druhým stupněm Burrowsova-Wheelerova kompresního algoritmu. Typickým představitelem druhého stupně je „přesuň na začátek“ (MTF). MTF a další představitelé transformace globální struktury jsou popsáni v kapitole 3.5. Transformace globální struktury se snaží změnit pravděpodobnost výskytu jednotlivých znaků z lokálních míst na globální. Z výstupu Burrowsovy-Wheelerovy transformace dostane transformace globální struktury text, který obsahuje sledy stejných znaků s velkou lokální pravděpodobností. Typické je, že se tyto sledy se často mění, a zároveň se mění i pravděpodobnost výskytu jiných znaků.

3.3 Kódování délkou sledů

Kódování délkou sledů je jednoduché a bezeztrátové. Díky své jednoduchosti je hodně populární. Hlavní myšlenkou je nahrazování dlouhých posloupností stejných symbolů kratší posloupností znaků. Obvykle dlouhá posloupnost znaků je nahrazována kombinací speciálního symbolu, dvojice znaků a počtu opakování znaku. Uvedeme si to na příkladu.

Máme libovolný řetězec *aeecdbbbdhhhhhcccc* a jako speciální znak použijeme *@*. Aplikujeme kódování délkou sledů na tento řetězec a získáme nový řetězec *a@2ecd@3bd@5h@4c*

.

Z příkladu je zřejmé, že v některých případech může dojít k expanzi znaků. Proto můžeme připojit k kódování délkou sledů i určitý práh, při jehož dosažení bude posloupnost znaků nahrazena námi zvolenou posloupností. Pokud bychom znovu použili náš řetězec a práh si například zvolili 4, tak bychom získali řetězec v tomto tvaru *aeecdbbbd@5h@4c* .

kódování délkou sledů	vstupní řetězec	výstupní řetězec	délka
klasické	aeecdbbbdhhhhhcccc	a@2ecd@3bd@5h@4c	16
s prahem 4	aeecdbbbdhhhhhcccc	aeecdbbbd@5h@4c	15

Tabulka 3.3: Komprese dat za pomocí kódování délkou sledů

Z tabulky je vidět, že při použití druhé varianty je výsledná délka zakódováného řetězce o jeden znak kratší, ale při kompresi delších řetězců může být tento rozdíl mnohem větší.

Jinou variantou kódování délkou sledů je *kódování digramů*. Tato varianta je vhodná pro data, která obsahují pouze jisté znaky např. pouze písmena, číslice. Často se opakující dvojice znaků se zakódují do jednoho znaku. Pak je důležité znát tuto kompresní tabulku i při dekompresi. Zobecněním této metody je kódování n-gramů a nebo substituce celých slov.[7]

Inverzní funkce

Popíšeme si jednoduchý postup dekomprese kódování délky sledů. Procházíme zakódovaný text, pokud nenarazíme na speciální symbol @ tak znak, který načteme uložíme na výstup. Jakmile narazíme na speciální znak, načteme následující dva znaky. První znak určuje počet opakování a druhý znak, který se bude opakovat. Takto postupujeme až projdeme celý vstupní text.

3.4 Entropické kódování

Ve většině případů posledním blokem Burrowsova-Wheelerova algoritmu, který má výrazný vliv na kompresní poměr. Nejvýznamnější představitelé entropického kódování jsou popsáni v následující sekci.

3.4.1 Huffmanovo kódování

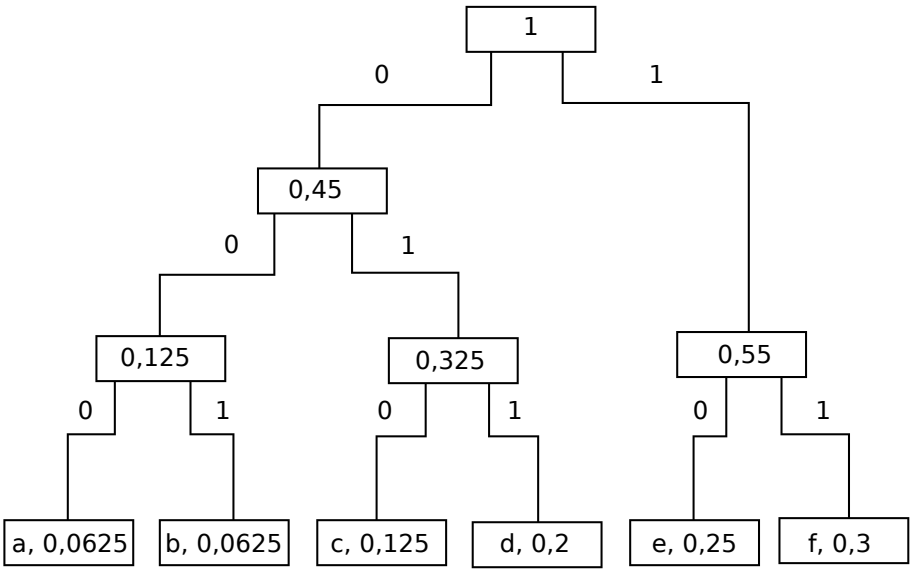
Huffmanovo kódování patří mezi bezeztrátové kompresní metody. Metoda je založena na znalosti pravděpodobnosti výskytu jednotlivých znaků. Existují různé modifikace získání pravděpodobnostních výskytů jednotlivých dat, mohou být získány ze statického, semiadaptivního nebo adaptivního modelu. Statické Huffmanovo kódování zná před zpracováním dat pravděpodobnosti výskytu jednotlivých znaků. Semiadaptivní metoda musí nejdříve data přečíst, aby si zjistila pravděpodobnosti výskytu znaků. Popisu adaptivního modelu se budeme věnovat později v části 3.4.2.

Získáme pravděpodobnosti výskytů jednotlivých znaků, které seřadíme vzestupně viz. tabulka 3.4. Ze seřazených dat zkonstruujeme Huffmanův strom. Začneme se znaky s nejnižší pravděpodobností výskytu. Vytvoříme z nich listy stromu a přiřadíme jim rodičovský uzel, který bude obsahovat hodnotu pravděpodobnosti součtu synovských uzlů. Dále budeme pokračovat podobně se zbylými znaky, vždy si vybereme ty s nejnižší pravděpodobností a

znak	a	b	c	d	e	f
pravděpodobnost výstkytu	0,0625	0,0625	0,125	0,2	0,25	0,3

Tabulka 3.4: Příklad tabulky frekvencí znaků

spojíme je do uzlu. Toto platí i pro vytvořené uzly. Skončíme, až se všechny uzly spojí do jednoho kořenového uzlu s pravděpodobností výskytu rovné jedna. Vytvořený Huffmanův strom můžeme vidět na obrázku 3.2. Jednotlivým hranám se přiřadí podle námi zvoleného pravidla hodnoty 1 nebo 0. V našem případě se pro pravé větve stromu zvolila 1 a pro levé 0. Z takto vytvořeného Huffmanova stromu můžeme jednoduše přiřadit jednotlivým



Obrázek 3.2: Vytvořený Huffmanův strom dle tabulky 3.4

znakům daný kód. Stačí jít od kořenového uzlu k příslušnému listu a pamatovat si posloupnost ohodnocených hran. Z tabulky 3.5 je vidět, že znakům s nejvyšší pravděpodobností

znak	pravděpodobnost	kód
a	0,0625	000
b	0,0625	001
c	0,125	010
d	0,2	011
e	0,25	10
f	0,3	11

Tabulka 3.5: Kódovací tabulka pro Huffmanovo kódování podle obrázku 3.2

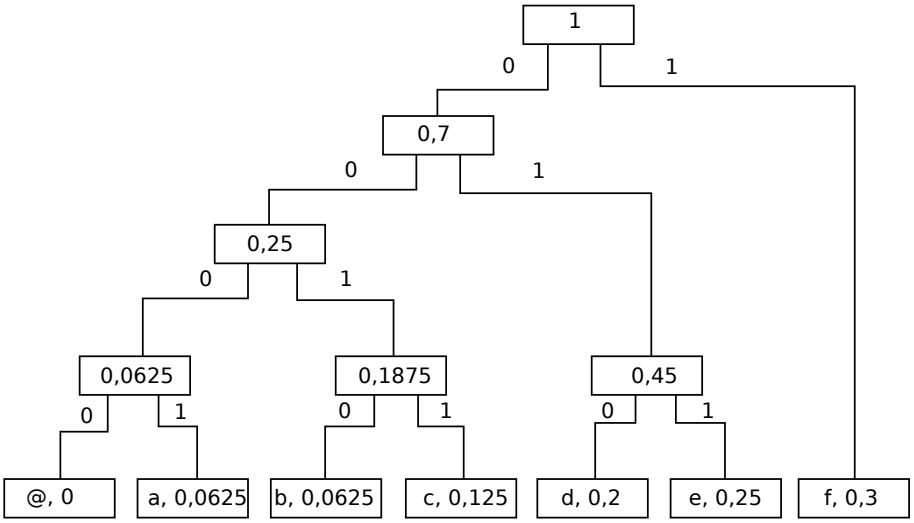
výskytu je přiřazen nejkratší kód. Se snižující se pravděpodobností výskytu se zvětšuje délka přiřazeného kódu.

Inverzní funkce

Proto, abychom mohli dekódovat text zakódovaný semiadaptivním Huffmanovým kódováním, potřebujeme znát tabulku frekvencí jednotlivých znaků, bez ní není možné text dekódovat. Proto si musíme tuto tabulku nebo jiná data, ze kterých tuto tabulku můžeme odvodit, uložit se zakódovanými daty. U dekódování postupujeme takto. Načteme si data (frekvence jednotlivých znaků, počet výskytů jednotlivých znaků…), ze kterých si vytvoříme Huffmanův strom. Pak již jen načítáme ze vstupního textu po bitech a procházíme Huffmanovým stromem. Když se dostaneme k listu, zapíšeme do výstupního textu znak, který je v tomto listu obsažen a vrátíme se zpět do kořenového uzlu. Tento postup opakujeme, dokud nenačteme celý vstupní text.

3.4.2 Adaptivní Huffmanovo kódování

Tato metoda na začátku nezná pravděpodobnostní výskyty jednotlivých znaků. A tak začíná s prázdným Huffmanovým stromem. V průběhu čtení dat se mění jak pravděpodobnost znaků, tak se i upravuje Huffmanův strom. Načte se znak, vloží se jeho příslušný kód na výstup. Pokud se tento znak dosud nevyskytl mezi načtenými znaky, je vložen na výstup v podobě, jaké byl načten. Poté je aktualizována pravděpodobnost znaku a upraven Huffmanův strom. Tento postup zaručuje reverzibilnost. Pokud se při každém načteném znaku přeuspořádává Huffmanův strom, může to být časově náročné. Tuto operaci můžeme provést i po načtení několika znaků. Je důležité, aby toto věděl i dekodér. Je tu však problém s rozpoznáním komprimovaného kódu od vloženého nekomprimovaného znaku. Proto se zavádí speciální symbol, který je vždy vložen před nekomprimovaný znak. Tento speciální symbol je vložen do stromu s nulovou frekvencí výskytu, která se nemění. Pokud bychom využili již sestavený Huffmanův strom z obrázku 3.2 a vložili do něho tento speciální znak, pak by strom vypadal jako na obrázku 3.3.



Obrázek 3.3: Huffmanův strom pro adaptivní Huffmanovo kódování

3.4.3 Aritmetické kódování

Aritmetické kódování je bezeztrátové kódování [4]. Pracuje podobně jako Huffmanovo kódování, které dává dobré výsledky v případě, že pravděpodobnost jednotlivých znaků je mocninou 2. Aritmetické kódování nepřiřazuje kód jednotlivým symbolům vstupního řetězce, ale celému vstupnímu textu. Kódové slovo leží na intervalu $0{,}0 \leq n < 1{,}0$. Aritmetické kódování bez modifikací pracuje dvouprůchodově. V prvním průchodu zjistí pravděpodobnosti symbolů vstupní abecedy. Ve druhém průchodu tvoří kódové slovo. Postup kódování si ukážeme na příkladě.

Při prvním průchodu zjistíme pravděpodobnostní model dat. Vezmeme si pro náš příklad slovo *fotografie*. Jednotlivým znakům bude třeba spočítat pravděpodobnost jejich výskytu. Interval $0{,}0 \leq n < 1{,}0$ se rozdělí úměrně podle pravděpodobnosti výskytu znaků.

znak	počet výskytu	pravděpodobnost	interval	kumulovaná frekvence
f	2	2/10 = 0,2	<0,8; 1,0)	0
o	2	2/10 = 0,2	<0,6; 0,8)	2
a	1	1/10 = 0,1	<0,5; 0,6)	4
e	1	1/10 = 0,1	<0,4; 0,5)	5
g	1	1/10 = 0,1	<0,3; 0,4)	6
i	1	1/10 = 0,1	<0,2; 0,3)	7
r	1	1/10 = 0,1	<0,1; 0,2)	8
t	1	1/10 = 0,1	<0,0; 0,1)	9
Celková kumulovaná frekvence				10

Tabulka 3.6: Tabulka frekvencí a pravděpodobnosti znaků

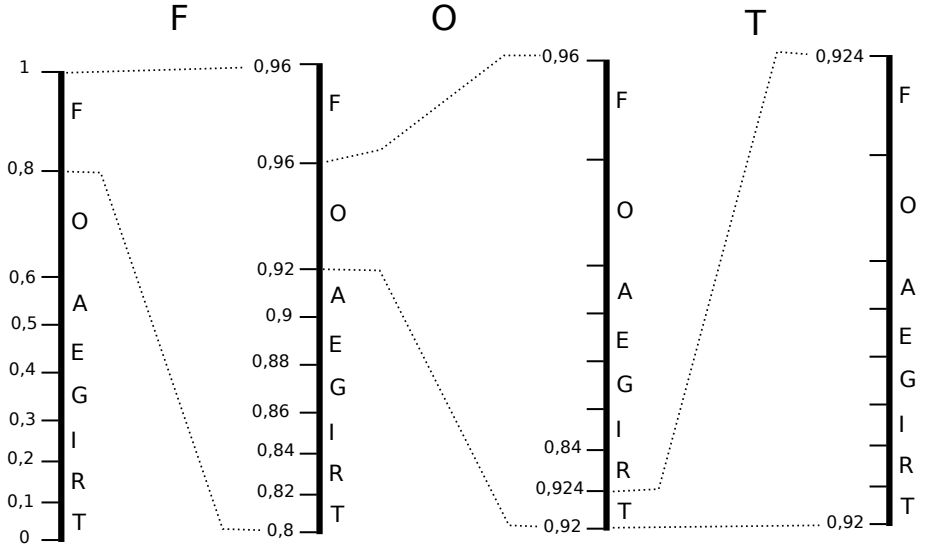
Po sestavení intervalů pro jednotlivé znaky můžeme začít zpracovávat vstupní text. Načteme první znak, v našem případě je to písmeno f , které leží na podintervalu <0,8; 1). Tento podinterval vezmeme jako rozsah kódového slova a rozdělíme ho opět podle pravděpodobnosti výskytu jednotlivých znaků. Tento postup je znázorněn na obrázku 3.4. Takto pokračujeme, dokud nenačteme všechny znaky ze vstupního textu. Získáme interval, ve kterém jakékoli číslo reprezentuje náš vstupní text. Problémem může být reprezentace tohoto desetinného čísla. Proto je možné upravit aritmetický kóder, aby využíval jen celá čísla.

3.4.4 Modifikace aritmetického kódování

Adaptivní aritmetické kódování nastaví v pravděpodobnostním modelu výskyty jednotlivých znaků na stejnou hodnotu. Nejdříve se zpracuje načtený symbol, a až poté se aktualizuje pravděpodobnostní model. Toto je důležité, aby byl proces reverzibilní. Pravděpodobnostní model musí být udržován v seřazeném pořadí.[7, 10]

Jelikož aritmetické operace s desetinnými čísly obnášejí větší režii, je vhodné pracovat s celými čísly. Ale i práce s celými čísly není dostačující, protože jsme omezeni na určitý konečný interval. Proto se využívá škálování. Jednoduše můžeme zjistit, v jaké části intervalu se nacházíme a podle toho provést změny měřítka intervalu. Existuje několik variant škálování např.: E2, E3. Jednotlivé varianty se liší podle počtu podintervalů, na které jsou děleny. Zde bude popsána varianta E3, která využívá následující tři podintervaly. Příklad si ukážeme na realných číslech na intervalu $< 0; 1)$.

1. E1 - podinterval je zcela obsažen v dolní polovině intervalu (je v intervalu <0;0,5))



Obrázek 3.4: Dělení intervalů u aritmetického kódování

2. E2 - podinterval je zcela obsažen v horní polovině intervalu (je v intervalu <0,5;1))
3. E3 - podinterval se nachází ve středu intervalu (je v intervalu <0,25;0,75))

Jakmile se podinterval zúží natolik, že je obsažen v jednom z těchto intervalů, dojde ke změně měřítka, ale u každého podintervalu jinak. Jsme-li v podintervalu E1, tak dojde k vynásobení dolní i horní hranice dvěmi. Pokud jsme v podintervalu E2, dojde k odečtení poloviny šířky intervalu k dolní a horní hranici (v našem případě to je 0,5) a vynásobení dvěmi. V případě třetího podintervalu dochází k odečtení jedné čtvrtiny celého intervalu (v našem případě 0,25) a vynásobení dolní i horní hranice dvěmi.

Pokud se nacházíme v E1 nebo E2, tak nejvýznamější bit má hodnotu 0 nebo 1 tj. nejvýznamější bit dolní i horní hranice se shodují, tak tento bit zapisujeme na výstup. Díky této vlastnosti se může dekodér orientovat, ve které části intervalu se kódové slovo nachází.

Tento algoritmus by se dal popsat pomocí pseudokódu takto:

```
while(1)
{
  // E1 podinterval
  if(horniMez < POLOVINA_INTERVALU)
  {
       ZapisBitNaVystup(0);
       while(citac > 0)
       {
         ZapisBitNaVystup(1);
         citac--;
       }
  }
  // E2 podinterval
  else if (low => POLOVINA_INTERVALU)
  {
       ZapisBitNaVystup(1);
       while(citac > 0)
       {
```

```
        ZapisBitNaVystup(0);
        citac--;
      }

      dolniMez -= POLOVINA_INTERVALU;
      horniMez -= POLOVINA_INTERVALU;

  }
  // E3 podinterval
 else if ((dolniMez >= PRVNI_TRETINA_INTERVALU)
              && (horniMez < TRETI_TRETINA_INTERVALU) )
  {
      dolniMez -= PRVNI_TRETINA_INTERVALU;
      horniMez -= PRVNI_TRETINA_INTERVALU;
      citac++;
  }
  else
  {
    break;
  }
  dolniMez = dolniMez * 2;
  horniMez = horniMez * 2 + 1;
}
```

Pseudokód 3.3: Komprese pomocí aritmetického kódování

Proto, aby bylo řádně ukončeno kódování poslední části vstupního řetězce, je nutné zapsat na výstup i zbylé nezapsané bity. Musíme zaručit, aby poslední kódovaný znak ležel ve správném podintervalu. Toto docílíme následujícím kódem.

```
if(dolniMez < PRVNI_TRETINA_INTERVALU)
  {
    ZapisBitNaVystup(0);

    for (i = 0; i < citac; i++)
    {
      ZapisBitNaVystup(1);
    }
  }
  else
  {
    ZapisBitNaVystup(1);

    for (i = 0; i < citac; i++)
    {
      ZapisBitNaVystup(0);
    }
  }
```

Pseudokód 3.4: Dokončení komprese pomocí aritmetického kódování

Pro dekódování použijeme podobný postup jako při kódování, které si ukážeme na následujícím algoritmu. Budeme si muset udržovat hodnotu kódového slova *value*.

```
while(1)
```

```
{
    if(horniMez < POLOVINA_INTERVALU)
    {
    }
    else if (dolniMez >= POLOVINA_INTERVALU)
    {
        dolniMez -= POLOVINA_INTERVALU;
        hodnota -= POLOVINA_INTERVALU;
        horniMez -= POLOVINA_INTERVALU;
    }
    else if ((low >= PRVNI_TRETINA_INTERVALU)
                    &&(high < TRETI_TRETINA_INTERVALU) )
    {
        dolniMez -= PRVNI_TRETINA_INTERVALU;
        hodnota -= PRVNI_TRETINA_INTERVALU;
        horniMez -= PRVNI_TRETINA_INTERVALU;
    }
    else
    {
        break;
    }

    dolniMez = dolniMez * 2;
    horniMez = horniMez * 2 + 1;
    hodnota = hodnota * 2 + PrectiBitZeVstupu();
}
```

Pseudokód 3.5: Dekomprese pomocí aritmetického kódování

3.4.5 Riceovo-Golombovo kódování

Riceovo-Golombovo kódování je kódování s variabilní délkou [9]. O proti Huffmanově kódování, které je založeno na pravděpodobnostním modelu dat, je Riceovo-Golombovo kódování založeno na hodnotě dat. Riceovo-Golombovo kódování je optimální pro vstupní data, která mají geometrické rozložení dat (kladných integer hodnot).

Základní popis

Je vhodné pro vstupní kódová slova, která mají exponenciální rozložení výskytů symbolů. Riceovo-Golombovo kódování pracuje s těmito konstantami nastavitelný dělitel M, kvocient Q a zbytek R. Vztah mezi hodnotou symbolu S ze vstupního řetězce a zmíněnými konstantami je vyjádřen rovnicí 3.2.

$$S = Q \cdot M + R \tag{3.2}$$

$$R = S \ MOD \ M \tag{3.3}$$

S pomocí rovnice 3.2 spočítáme proměnné Q a R. Tyto dvě proměnné jsou výstupními hodnotami kódovaného symbolu S. Kvocient se na výstup zapíše jako posloupnost jedniček o délce hodnoty kvocientu zakončené nulou. Zbytek je vyjádřen rovnicí 3.4, kde operátor MOD značí modulo, a na výstup zapsán bez jákékoliv úpravy. Pouze je vyjádřen na počet bitů, který lze vyjádřit proměnná $Q - 1$. Postup kódování si ukážeme pomocí jednoduché tabulky 3.7, kde proměnná $M = 4$.

S	Q	Q výstup	zbytek	zbytek výstup	výstup
0	0	0	0	00	0 00
1	0	0	1	01	0 01
2	0	0	2	10	0 10
3	0	0	3	11	0 11
4	1	10	0	00	10 00
5	1	10	1	01	10 01
6	1	10	2	10	10 10
7	1	10	3	11	10 11
8	2	110	0	00	110 00
9	2	110	1	01	110 01
10	2	110	2	10	110 10
11	2	110	3	11	110 11
12	3	1110	0	00	1110 00
13	3	1110	1	01	1110 01
14	3	1110	2	10	1110 10
15	3	1110	3	11	1110 11
16	4	11110	0	00	11110 00

Tabulka 3.7: Kódování dat pomocí Riceovo-Golombovo kódování

Z příkladu v tabulce 3.7 vidíme, že v případě hodnoty 255 bychom měli 15 jedniček jako kvocient a pokud připočteme separátor 0 a zbytek, dostaneme se na délku 20 bitů. Proto je důležité vhodně zvolit dělitel M.

Exponenciální Riceovo-Golombovo kódování

Exponenciální Riceovo-Golombovo kódování se vypočítá podle vzorce 3.4.

$$S = (2^Q) - 1 + R \tag{3.4}$$

Kvocient udává, na kolika bitech bude vyjádřen zbytek. Pokud použijeme tabulku 3.7 a upravíme ji, získáme následující výsledky.

Z tabulky 3.8 vidíme, že pokud bude mít vstupní kódové slovo exponenciální rozložení znaků, tak docílíme vysokého kompresního poměru. Ale pokud vstupní data nebudou mít geometrické rozložení, tak výsledný komprimovaný soubor nebude dosahovat nejlepšího kompresního poměru.

3.5 Modifikace transformace globální struktury

V této kapitole si popíšeme druhou část Burrowsova-Wheelerova algoritmu [6, 1]. V této kapitole budou popsány různé metody, které se mohou být na druhém místě v Burrowsově-Wheelerově algoritmu. U některých metod budou popsány i možnosti jejich modifikací.

3.5.1 Přesuň na začátek

Mezi nejznámější představitele druhého bloku BWCA patří metoda přesuň na začátek (move to front). Je použita v mnoha implementacích BWCA. Samotná metoda nekom-

S	Q	Q výstup	zbytek	zbytek výstup	výstup
0	0	0	0	0	0
1	1	10	1	0	10 0
2	1	10	1	1	10 1
3	2	110	2	00	110 00
4	2	110	2	01	110 01
5	2	110	2	10	110 10
6	2	110	2	11	110 11
7	3	1110	3	000	1110 000
8	3	1110	3	001	1110 001
9	3	1110	3	010	1110 010
10	3	1110	3	011	1110 011
11	3	1110	3	100	1110 100
12	3	1110	3	101	1110 101
13	3	1110	3	110	1110 110
14	3	1110	3	111	1110 111
15	4	11110	4	0000	11110 0000
16	4	11110	4	0001	11110 0001

Tabulka 3.8: Ukázka komprese dat pomocí exponenciálního Riceovo-Golombovo kódování

primuje data jen se snaží snížit redundanci. Pouze přetrasnformovává výstupní data z Burrowsovy-Wheelerovy metody.

Základní popis

Přesuň na začátek transformace dále jen MTF je obvykle používána po provedení Burrowsovy-Wheelerovy transformace. MTF si udržuje abecedu vstupních symbolů v seznamu. Pracuje proudově a je reverzibilní. Její základní myšlenkou je transformace sekvence vstupních symbolů na sekvenci jejich indexů v abecedním seznamu.

Na začátku transformace je seznam seřazen v abecedním pořadí. Při každém načtení znaku ze vstupu se znak vyhledá v seznamu a jeho index určující pozici se zapíše na výstup. Poté se znak přesune na první pozici v seznamu. Názorný příklad je uveden v tabulce 3.9. Tento postup zaručí, že často se opakující symboly budou uloženy na začátku seznamu a na výstup se bude generovat více znaků *0*.

Mějme vstupní abecedu A =$\{a, b, c, d, e, f, g\}$ a vstupní tok *baadcaadaagef* tak transformace bude probíhat takto:

Pokud vstupní tok obsahuje koncentrace identických symbolů (nazýva se *"koncentrační vlastnost"*), tak MTF dává dobré výsledky. Sekvence opakujících se symbolů o délce m bude zapsána na výstup jako posloupnost *m-1* znaku *0*. Je to dáno tím, že při prvním načtení symbolu X se na výstup zapíše jeho aktuální index, a až potom je přesunut na první pozici v seznamu.

Inverzní funkce

V tomto odstavci si popíšeme inverzní funkci k MTF. Tuto funkci je možné použít pro dekódování MTF-1 a MTF-2 s drobnými úpravami. Nejdříve si musíme nainicializovat počáteční pole znaků. Je důležité, aby bylo stejné jako to, se kterým jsme zakódovávali text. Pak již

Vstupní symbol	Seznam	Výstupní index
b	abcdefg	1
a	bacdefg	1
a	abcdefg	0
d	abcdefg	3
c	dabcefg	3
a	cdabefg	2
a	acdbefg	0
d	acdbefg	2
a	dacbefg	1
a	adcbefg	0
g	adcbefg	0
e	adcbefg	4
f	eadcbfg	0

Tabulka 3.9: Kódování za pomocí transformace Přesuň na začátek

jen stačí načíst znak ze vstupního textu, vyhledat příslušnou pozici v poli znaků. Znak uložený na této pozici uložíme do výstupního textu. Nalezený znak přesuneme na první místo a ostatní znaky posuneme. Takto pokračujeme, až zpracujeme celý vstupní text.

Modifikace

Popíši zde dvě metody označované MTF-1 algoritmus od Balkenhola a MTF-2 algoritmus od Balkenhola a Starkova. Původní MTF transofmace přesune symbol na začátek ihned po jeho načtení. MTF-1 a MTF-2 algoritmy produkují více nul na výstupu oproti nemodifikované verzi.

Transformace MTF-1 dovolí přesunout na první pozici v seznamu jen znak, který je na druhé pozici. Je-li přesouván znak z vyšší pozice je přesunut na druhou pozici.

Transformace MTF-2 pracuje obdobně jen s tím rozdílem, že znak z druhé pozice přesouvá na první pouze tehdy, pokud nebyl v předchozím kroku použit symbol z první pozice. Obě transformace pomaleji reagují na výskyt nových symbolů.[6, 3]

3.5.2 Distanční kódování

Distanční kódování (distance coding zkráceně DC) je algoritmus navržený Edgarem Binderem v roce 2000. DC je náhradou za MTF v Burrowsově-Wheelerově kompresním algoritmu. Tento algoritmus vyhledává ve vstupním algoritmu výskyty stejných znaků. Do výstupního souboru pouze zapisuje vzdálenost mezi výskyty dvou stejných znaků. Při dekódování známe poslední pozici daného znaku a tak lehce dopočítáme jeho novou pozici.

Základní popis

V prvním kroku nalezneme všechny první výskyty jednotlivých znaků ve vstupním textu a zapamatujeme si jejich pozice. Jakmile jsme zjistili všechny první výskyty znaků, tuto posloupnost zapíšeme do výstupního textu. V následujícím kroku hledáme druhé výskyty těchto znaků, ale zapamatujeme si vzdálenost (počet znaků mezi výskyty dvou stejných znaků) mezi těmito znaky. Jsou-li všechny druhé výskyty jednotlivých znaků zpracovány,

zapíšeme zapamatované vzdalenosti do výstupního souboru a pokračujeme druhým krokem. Tento postup si ukážeme na jednoduchém příkladu.

Máme vstupní abecedu obsahující znaky *a, b, c, d, r* a vstupní text *abracadabra*. Začneme tím, že nalezneme pro každý znak vstupní abecedy jeho první výskyt ve vstupní řetězci viz. tabulka 3.10.

Znak	a	b	c	d	r
Pozice	1	2	5	7	3

Tabulka 3.10: První krok distančního kódování

Do výstupního souboru bude zápsána tato posloupnost pozic *1 2 5 7 3*. Poté bude prováděn druhý krok, dokud nebude zpracován celý vstupní text. Při každém kroku bude vyhledána pozice dalšího výskytu všech znaků viz. 3.11.

Znak	a	b	c	d	r
Pozice	3	7	0	0	7
Znak	a	b	c	d	r
Pozice	3	0	0	0	0
Znak	a	b	c	d	r
Pozice	2	0	0	0	0
Znak	a	b	c	d	r
Pozice	3	0	0	0	0

Tabulka 3.11: Iterace druhých kroků distančního kódování

Po zpracování celého vstupního textu bude ve výstupním textu tato posloupnost *1 2 5 7 3 3 7 0 0 7 3 0 0 0 0 0 2 0 0 0 0 0 3 0 0 0 0 0*.

Inverzní funkce

Pro dekódování distančního kódování budeme potřebovat vytvořit pole, do kterého budeme zapisovat jednotlivé znaky. Zároveň musíme znát pořadí jednotlivých znaků vstupní abecedy, podle kterého byly uloženy hodnoty vzdáleností jednotlivých znaků. Vždy načteme počet vzdáleností, jako je počet znaků vstupní abecedy. Znak zapíšeme na pozici získanou součtem pozice posledního výskytu daného znaku a načtené vzdálenosti. V prvním kroku je pozice posledního výskytu znaku nastavena na 0. Tento postup si ukážeme na příkladu.

Mějme vstupní text *1 2 5 7 3 3 7 0 0 7 3 0 0 0 0 0 2 0 0 0 0 0 3 0 0 0 0 0*. Načteme stejný počet hodnot vstupního textu, jako je počet znaků vstupní abecedy.

Znak	a	b	c	d	r
Aktuální pozice	0	0	0	0	0
Vzdálenost	1	2	5	7	3
Následující pozice	1	2	5	7	3

Tabulka 3.12: První krok inverzní funkce distančního kódování

Z tabulky 3.12 vidíme, že v prvním kroku je nastavena aktuální vzdálenost všech znaků na 0. K aktuální pozici se přičte načtená vzdálenost a na takto vypočítanou pozici se vloží daný znak. V tabulce 3.13 je ukázáno pole, které představuje výstupní text.

Index	1	2	3	4	5	6	7	8	9	10	11
Znak	a	b	r	-	c	-	d	-	-	-	-

Tabulka 3.13: Výstupní text inverzní funkce distančního kódování

Pokračujeme opět stejným způsobem, načteme hodnoty ze vstupního textu, přiřadíme znakům vstupní abecedy a vypočítáme jejich následující pozici.

V tabulce 3.14 byla některým znakům načtena vzdálenost 0. Tyto znaky již nebudou přiřazovány na nové pozice. Po tomto kroku bude vypadat pole představující výstupní text následovně.

Znak	a	b	c	d	r
Aktuální pozice	1	2	5	7	3
Vzdálenost	3	7	0	0	7
Následující pozice	4	9	5	7	10

Tabulka 3.14: Opakovaný krok inverzní funkce distančního kódování

Index	1	2	3	4	5	6	7	8	9	10	11
Znak	a	b	r	a	c	-	d	-	b	r	-

Tabulka 3.15: Výstupní text inverzní funkce distančního kódování

Následné kroky by byly stejně zpracovány jako uvedený předcházející krok. Po načtení všech hodnot ze vstupního textu získáme výstupní text *abracadabra*, který odpovídá textu, který jsme na začátku této kapitoly zakódovali.

Modifikace

Možné modifikace u této metody mohou být podobné jako u inverzní frekvence 3.5.3. Další možnou modifikací je do výstupního textu zapisovat jen první výskyt nulové vzdálenosti u daného znaku a tento znak vyřadit z následujícího zpracování vstupního textu.

3.5.3 Iverzní frekvence

Stejně jako distanční kódování je metoda inverzní frekvence (inversion frequencies zkráceně IF) založena na měření vzdáleností výskytů znaků. Algoritmus byl navržen Z. Arnavutem a S. Magliverasem. Algoritmus dává lepší výsledky téměř u všech souborů než MTF.

Základní popis

Máme vstupní řetězec x^{bwt} a abecedu vstupních symbolů A_{in}. Pro každý symbol $a \in A_{in}$ se hledá shoda ve vstupním řetězci x^{bwt}. Při prvním nalezeném znaku a_j se zapíše na výstup x^{if} pozice tohoto znaku ve vstupním řetězci x^{bwt}. U následujících shod se postupuje tak, že na výstup x^{if} se vloží číslo, které určuje počet znaků větších než aktuální znak a_j na intervalu od posledního výskytu po aktuální. Jednou z výhod IF je, že pro poslední znak abecedy (z) není potřeba hledat jeho výskyty ve vstupním řetězci x^{bwt}, protože by obsahovala samé 0. Díky tomu je i výstupní řetězec x^{if} menší než vstupní řetězec x^{bwt}.

K obnovení vstupního řetězce x^{bwt} z výstupního řetězce x^{in} je potřeba znát frekvence znaků abecedy nebo ukončující symbol za každou výstupní posloupností hledaného znaku a_j. Bez těchto informací neobnovíme korektně x^{bwt}.

Modifikace

Pokud bychom nejdříve zpracovali znaky abecedy s vyšší pravděpodobností a pokračovali těmi s nižší, zpracované znaky s nižší frekvencí by měly menší hodnoty na výstupu. Nevýhoda tohoto zpracování je, že znaky s vysokou pravděpodobností budou delší než znaky s menší pravděpodobností. Z toho důvodu se využívá také opačného postupu tj. nejdříve se zpracují znaky s menší pravděpodobností.[6, 3] Tato metoda se nazývá řazená inverzní

frekvence (SIF). Je otázkou, zda zvolit vzestupné nebo sestupné seřazení. Určitým vodítkem může být tento postup. Pomocí následujících rovnic si vypočítáme parametr, podle kterého se budeme řídit při určení seřazení. Pro každý znak a ze vstupním abecedy A_{in} bude f_a představovat počet výskytů znaku a ve vstupním řetězci X_{in}. Proměnná F_{avg} představuje průměrný počet znaků ve vstupním textu X_{in} o délce n.

$$F_{avg} = \frac{n}{|A_{in}|} \tag{3.5}$$

Proměnná G je množina symbolů jejichž f_a je větší nebo rovna $2F_{avg}$.

$$G = \{a|f_a > 2F_{avg}|\} \tag{3.6}$$

Proměnná S představuje procentuální poměr symbolů a ve vstupní abecedě A_{in}, pro které je f_a větší nebo rovno $2F_{avg}$.

$$S = 100\frac{|G|}{|A_{in}|} \tag{3.7}$$

Podle takto získaných hodnot musíme určit práh, podle kterého se budeme řídit při určování vzestupného nebo sestupného seřazení.

3.5.4 Vážený frekvenční počet

Vážený frekvenční počet (weighted frequency count zkráceně WFC) pracuje podobně jako transformace přesuň na začátek. Tento algoritmus byl prezentován S. Deorowiczem v roce 2001 [6]. WFC vypočítává pro každý symbol vstupní abecedy jeho váhovou hodnotu. Váhová hodnota se určuje pro každý načtený symbol a s větší vzdáleností od aktuálně načteného znaku klesá. Díky této vlastnosti algoritmus má dobrý kompresní poměr, ale na úkor výpočetnímu výkonu, který je nutný k výpočtu váhové hodnoty jednotlivých symbolů.

Základní popis

WFC stejně jako MTF nahrazuje vstupní symbol x odpovídající hodnotou funkce $y(x)$. Rozdíl je právě ve funkci $y(x)$. Výstupem funkce v MTF je aktuální pozice v seznamu, která načte symbol x a přesune na začátek seznamu. MTF nebere v úvahu pravděpodobnost výskytu tohoto symbolu a dá ho hned na začátek. Tím může posunout znaky s vyšší pravděpodobností stranou. Toto vede ke zhoršení kompresního poměru. Funkce $y(x)$ ve WFC pracuje s posuvným oknem o velikosti t_{max}. Obsahuje pro každý symbol jeho váhu, podle které je seřazeno. Symboly s vyšší váhou jsou na nejnižších pozicích. Nejnižší pozice je 0. Podle vzdálenosti dvou stejných symbolů je odvozována jeho váha. Symboly, které jsou blízko u sebe, mají větší váhu než ty, které mají mezi sebou větší vzdálenost. Stejně jako u MTF se na výstup vkládá pozice v posuvném okně.

Modifikace

Záleží na velikosti posuvného okna. Větší okno dává lepší kompresní výsledky, ale je zapotřebí mnohem více výpočtů, které zpomalují algoritmus. Nejlepší kompresní výsledky jsou dosaženy u 12. logaritmické úrovně. Pro velikost t_{max} to znamená, že je použita hodnota 2048. Váhová funkce, která se vrací stejná pro všechny soubory, nemusí mít optimální kompresní poměr. Soubor od souboru má rozdílnou souborovou strukturu. Pro některé soubory

je vhodné použít váhovou funkci, která klade větší důraz na právě načtené symboly, než na symboly starší nebo naopak. Velice důležité je rozložení symbolů v souboru. Nyní si definujme rovnice, které nám pomohou k určení váhové funkce. Hodnota f_a udává počet výskytů symbolu a ve vstupním textu. Hodnota n je délka vstupního textu. A_{in} představuje pravděpodobnostní rozložení jednotlivých symbolů vstupní abecedy. První rovnice vrací průměrnou frekvenci vstupního textu s délkou n.

Funkce $f(l)$ s parametry p_0, p_1 a S:

$$f_{p_0,p_1,S^{(l)}} = \begin{cases} 2^{17}, & \text{l=0} \\ 2^{14}, & \text{l=1} \\ \frac{f_{p_0,p_1,S^{(l-1)} \cdot p_0}}{p_1+(l \cdot S^2)}, & l \geq 2 \end{cases} \tag{3.8}$$

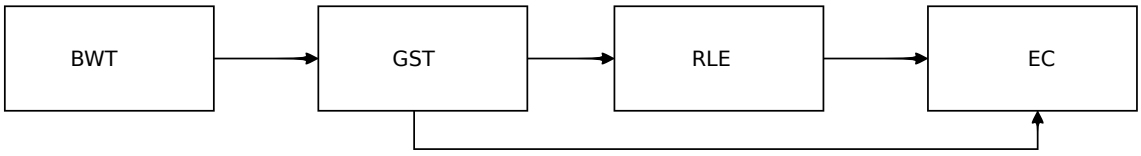
Vzdálenost aktuální pozice v posuvném okně je popsána hodnotou t. Začíná od nuly jako následující levý symbol od aktuální pozice. Pak váhová funkce je definována $w_{p_0,p_1,S^{(t)}}$ takto:

$$f_{p_0,p_1,S^{(l)}} = \begin{cases} p_0, p_1, S^{(0)}, & t = 0, \\ p_0, p_1, S^{(1)}, & 2^0 \leq t \leq 2^1 - 1, \\ p_0, p_1, S^{(2)}, & 2^1 \leq t \leq 2^2 - 1, \\ p_0, p_1, S^{(3)}, & 2^2 \leq t \leq 2^3 - 1, \\ \vdots & \\ p_0, p_1, S^{(11)}, & 2^{10} \leq t \leq 2^{11} - 1, \\ 0, & t \geq 2048, \end{cases} \tag{3.9}$$

Záleží na vhodné kombinaci p_0, p_1 pro jednotlivé soubory, protože výsledky se mohou lišit soubor od souboru.

3.5.5 Inkrementační frekvenční počet

Inkrementační frekvenční počet (incremental frequency count zkráceně IFC) pracuje na stejném principu jako WFC [2]. WFC dává dobrý kompresní poměr, ale za cenu vysoké výpočetní náročnosti spojené s počítáním vah jednotlivých symbolů. Při každém načtení symbolu se musely váhy u všech znaků přepočítávat. IFC se snaží dosáhnout stejného kompresního poměru jako WFC, ale s nižší složitostí. Toho dosahuje IFC tak, že v posuvném okně používá čítače pro jednotlivé symboly. Při načtení symbolu aktualizuje jen čítač tohoto symbolu. Na obrázku 3.5 je vidět, že je nutné prohodit druhý a třetí blok BWCA. Kódování délkou sledů zasílá délku sledu l přímo do EC.



Obrázek 3.5: Burrowsův-Wheelerův kompresní algoritmus

Základní popis

IFC obsahuje pět částí. Jednotlivé čítače jsou seřazeny v sestupném pořadí. Na začátku je důležité všechny čítače vynulovat. Jednotlivé části IFC:

1. Načítá znaky ze vstupního textu a na výstup dává index čítače daného znaku.
2. Počítá rozdíl mezi dvěma indexy průměru. Velikost posuvného okna je *window_size*. Index průměru avg_i na pozici i ve vstupním textu s aktuálním indexem $index_i$ se spočítá.
$$avg_i = \frac{(avg_{i-1} \cdot (window\_size - 1)) + index_i}{window\_size} \tag{3.10}$$
3. Inkrementuje čítač o hodnotu inc_i, která se spočítá takto:
$$dif_i = avg_i - avg_{i-1} \tag{3.11}$$
Hodnota maxima dm je pevně nastavena. Funkce Min vrátí menší hodnotu ze dvou vstupních parametrů.
$$difl_i = Min(|dif_i|, dm) \cdot dif_i \tag{3.12}$$
$$inc_i = inc_{i-1} - (\frac{inc_{i-1} \cdot difl_i}{64}) \tag{3.13}$$
4. Tato část hlídá jednotlivé čítače. Jakmile některý z nich dosáhne hodnoty 256, tak všechny čítače a inc_i jsou děleny dvěma.
5. Řadí seznam čítačů v sestupném pořadí.

Modifikace

Možností jak modifikovat tuto metodu je přepínání kontextu u EC. Víme, že pokud dostaneme na výstupu z IFC hodnotu 0, bude následovat nenulová hodnota a právě v tomto okamžiku dojde k přepnutí. Právě přepnutí kontextu může vést ke zlepšení kompresního poměru.[3]

3.5.6 Intervalové kódování

Intervalové kódování (interval encoding zkráceně IE) bylo představeno ve stejné době jako MTF. Tuto metodu popsal Peter Elias. [8]. IE je jednoduchá metoda, která pracuje obdobně jako Distanční kódování. Toto kódování, stejně jako předešlé, se snaží dávat na výstup co nejvíce 0,1.

Základní popis

Toto kódování lze popsat pomocí následujícího vzorce.

$$x(t) = Minimum\{k|\alpha(t-k) = \alpha(t)\} \tag{3.14}$$

Kde t označuje čas či pozici ve vstupním řetězci. Funkce α(t) vrací znak ze vstupní abecedy A, který je na pozici t vstupního řetězce. Proměnná k určuje vzdálenost mezi výskyty téhož znaku. Výstupem $x(t)$ na pozici t je tedy vzdálenost mezi aktuálně načteným znakem α(t) a jeho posledním výskytem α(t - k).

Pokud převedeme rovnici 3.14 do pseudokódu získáme tento kód s drobnými úpravami.

```
for(t = 1; t < N; t++)
{
  if (LAST[α(t)] == 0)
  {
    x(t) = t + α(t) - 1;
  }
  else
  {
    x(t) = t - LAST[α(t)];
  }

  LAST[α(t)] = t;
}
```

Pseudokód 3.6: Výpočet výstupní pozice v intervalovém kódování

Proměnná N se rovná délce vstupního řetězce. Pole $LAST$ je velké jako vstupní abeceda A udržuje v sobě poslední pozici pro každý symbol vstupní abecedy. Každý prvek tohoto pole je na začátku inicializován na hodnotu 0.

Kapitola 4

Testování

Tato kapitola obsahuje popis dat, na kterých se budou provádět testy. Druhá část této kapitoly se zaměřuje na testování metod transformace globální struktury. Jednotlivé metody, které budou testovány, byly popsány v kapitole 3.5. Měření se hlavně zaměřuje na kompresní výkon jednotlivých metod. Následně je porovnána i časová náročnost jednotlivých metod. Poté jsou vyhodnoceny jednotlivé metody a porovnány mezi sebou. Třetí část této kapitoly se věnuje testování entropických kodérů a jejich porovnání mezi sebou. Další testování se zaměřuje na možnost nastavení velikosti bloku dat, který bude řazen v Burrowsově-Wheelorově transformaci. V závěrečné části je porovnání mezi nejlepšími dosaženými výsledky s běžně používanými kompresními programy.

4.1 Testovací data

Proto, abychom mohli porovnávat jednotlivé kompresní, metody musí být zvolena vhodná data, na kterých bude prováděno testování. Toto nám zajišťují tak zvané *korpusy*. *Korpus* je sbírka přesně určených souborů, které slouží k porovnání výkonu rozdílných kompresních algoritmů. Soubory do *korpusu* jsou vybírány podle určitých pravidel. U vybraného souboru musí být zaručeno, že tento typ souboru se bude využívat i v budoucnosti. Velikost souboru by neměla být větší, než je nezbytně nutné. Soubor nesmí být chráněn autorským zákonem a musí být veřejně dostupný. Pokud soubor splňuje všechny tyto body může být vybrán do korpusu.

4.1.1 Calgary korpus

Jedním z prvních byl Calgary korpus. Byl vytvořen v roce 1987 Ianem Wittenem, Timothym Bellem a Johnem Clearym. Patří k nejznámějším korpusům, které se týkají komprese dat. Tento korpus slouží hlavně k porovnání kompresních metod zaměřujících se na kompresi textu, ale i celkově bezeztrátových kompresních metod. Calgary korpus existuje ve dvou variantách. První obsahuje 18 souborů a nazývá se velký Calgary korpus. Druhým je standardní Calgary korpus, který obsahuje nejpoužívanějších 14 souborů, jenž obsahuje velký Calgary korpus. Mezi obsažené typy dat patří anglicky psaný text, bibliografické citace, zdrojové kódy a obrázek. Jednotlivé soubory ze standardního Calgary korpusu si zde popíšeme.

Název	Velikost	Typ souboru	Popis souboru
bib	111 261 B	textový soubor	725 biblografických citací ve formátu refer.
book1	768 771 B	textové soubory	Kniha Thomase Hardyho *Far from the madding crowd*.
book2	610 856 B	textové soubory	Kniha ve formátu troff od Iana H. Wittena *Principles of computer speech*.
geo	102 400 B	binární soubor	Soubor obsahující geografická data.
news	377 109 B	textový soubor	Soubor s příspěvky z diskuzního fóra komunikačního systému Usenet.
obj1	21 504 B	binární soubor	Spustitelný program pro VAX.
obj2	246 814 B	binární soubor	Spustitelný program pro Macintosh.
paper1	53 161 B	textové soubory	Článek ve formátu troff od I. Wittena, R. Neala a J. Clearyho ıArithmetic coding for data compression.
paper2	82 199 B	textové soubory	Článek ve formátu troff od I. Wittena *Computer (in)security*.
pic	513 216 B	binární soubor	Obrázek o rozměrech 1728 x 2376.
progc	39 611 B	textové soubory	Zdrojový kód v programovacím jazyce C.
progl	71 646 B	textové soubory	Zdrojový kód v programovacím jazyce Lisp.
progp	49 379 B	textové soubory	Zdrojový kód v programovacím jazyce Pascal.
trans	93 695 B	textový soubor	Záznam terminálové relace.

Tabulka 4.1: Seznam souborů obsahující standardní Calgary korpus

4.1.2 Caterbury korpus

Korpus Caterbury je nástupce Calgary korpusu. Byl vyvinut v roce 1997 Rossem Arnolde a Timothy Bellem pro testování nových kompresních metod. Caterbury korpus je složen z 11 souborů, které byly vybrány z více než 800 souborů. Výběr souborů probíhal na základě jejich schopnosti poskytnout reprezentativní výsledky dle výkonu. V následující tabulce je výčet všech 11 souborů, které obsahuje Caterbury korpus 4.2.

Název	Velikost	Typ souboru	Popis souboru
alice29.txt	152 089 B	textový soubor	Kniha od Carroll Lewisové *Alice's adventures in wonderland.*
asyoulik.txt	125 179 B	textové soubory	Divadelní hra *As you like it* od Williama Shakespeara.
cp.htlm	24 603 B	textové soubory	Html stránka.
fields.c	11 150 B	textové soubory	Zdrojový kód programovacího jazyka C.
grammar.lsp	3 721 B	textové soubory	Zdrojový kód programovacího jazyka Lisp.
kennedy.xls	1 029 744 B	binární soubor	Dokument pro Excel.
lcet10.txt	426 754 B	textový soubor	Technická zpráva.
plrabn12.txt	481 861 B	textový soubor	Poezie, *Paradise lost* od Johna Miltona.
ptt5	513 216 B	binární soubor	Soubor testů pro CCITT.
sum	38 240 B	binární soubor	Spustitelný program pro SPARC.
xargs.1	4 227 B	textový soubor	GNU manuálová stránka.

Tabulka 4.2: Seznam souborů obsahující Caterbury korpus

4.1.3 Silecia korpus

Silecia korpus patří k nejnovějším korpusům. Byl sestaven Sebastianem Deorowiszem v roce 2003. Hlavním záměrem bylo pokrytí aktuálně nejpoužívanějších souborů. Silecia korpus se také snaží odstranit nevýhody předchozích dvou korpusů. Oproti předchozím dvěma korpusům obsahuje mnohem větší soubory o velikosti od 5 MB do 51 MB. Začleňuje i jiné textové soubory než jen ty, které jsou psány v anglickém jazyce. Proto byla vybrána kniha v polském jazyce. Dále je rozšířena o zdravotnické snímky a obrázky. Snaží se pokrýt i databázové soubory. Popis jednotlivých souborů nalezneme v následující tabulce 4.3.

Název	Velikost	Typ souboru	Popis souboru
dickens	10 192 446 B	textové soubory	Soubor knih Charlese Dickense.
mozilla	51 220 480 B	binární soubor	Spustitelný soubor programu Mozilla 1.0.
mr	9 970 564 B	binární soubor	Obrázek z magnetické rezonance.
nci	33 553 445 B	binární soubor	Databáze chemických struktur.
ooffice	6 152 192 B	binární soubor	DLL knihovna z Open Office.org 1.01.
osdb	10 085 684 B	binární soubor	Ukázková databáze v MySQL formátu.
reymont	6 627 202 B	binární soubor	Polská kniha *Chłopi* od Władysława Reymonta.
samba	21 606 400 B	textové soubory	Zdrojové kódy Samby 2-2.3.
sao	7 251 944 B	binární soubor	Astronomický katalog hvězd SAO.
webster	41 458 703 B	textové soubory	Websterův výkladový slovník.
xml	5 345 280 B	textové soubory	Sbírka XML souborů.
xray	8 474 240 B	binární soubor	Rengenový snímek.

Tabulka 4.3: Seznam souborů obsahující Silecia korpus

4.2 Porovnání kompresních metod

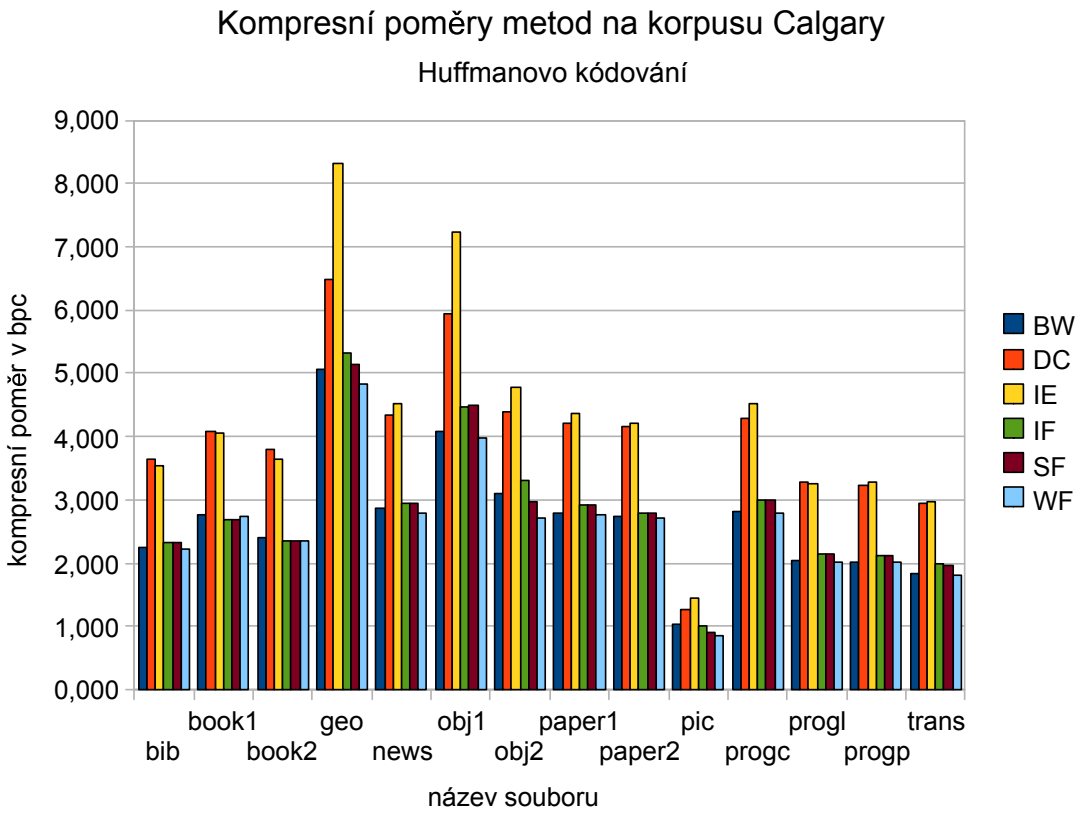
V této části budou testovány různé metody druhého stupně (transformace globální struktury) Burrowsovy-Wheelerovy transformace. Testy proběhnou na korpusech Calgary a Caterbury. U každého korpusu proběhnou dvě sady testů. Každý test proběhne s jiným entropickým kodérem. První bude s Huffmanovým kódováním a druhý s aritmetickým. Po těchto testech bude měřena časová náročnost jednotlivých metod. Na závěr je provedeno zhodnocení dosažených výsledků s přihlédnutím jak ke kompresnímu poměru tak i časové náročnosti. Údaje v tabulkách jsou dosažené poměry (v bpc).

4.2.1 Calgary korpus

Prvním testovaným korpusem je Calgary. Nejprve proběhnou testy s různými transformacemi globálních struktur (viz. kapitola 3.5) a entropickým kodérem jimž je Huffmanovo kódování (viz. kapitola 3.4.1). Jednotlivé testy se prováděly pomocí jednoduchého scriptu, který postupně spouštěl program pro jednotlivé soubory z Calgary korpusu. Výsledné kompresní poměry pro jednotlivé soubory z korpusu jsou v tabulce 4.4. Pro lepší znázornění byly data z tabulky vyvedeny do grafu 4.1.

Soubor	Velikost	MTF	DC	IE	IF	SIF	WFC
bib	111 261	2,234	3,646	3,534	2,326	2,326	**2,217**
book1	768 771	2,771	4,092	4,049	2,685	**2,685**	2,728
book2	610 856	2,414	3,809	3,648	2,360	**2,360**	2,360
geo	102 400	5,052	6,491	8,320	5,329	5,151	**4,839**
news	377 109	2,867	4,345	4,518	2,954	2,954	**2,781**
obj1	21 504	4,084	5,929	7,243	4,467	4,485	**3,981**
obj2	246 814	3,109	4,381	4,780	3,306	2,981	**2,715**
paper1	53 161	2,796	4,214	4,364	2,920	2,920	**2,776**
paper2	82 199	2,742	4,156	4,197	2,802	2,802	**2,700**
pic	513 216	1,026	1,275	1,446	1,006	0,896	**0,861**
progc	39 611	2,815	4,291	4,527	2,984	2,983	**2,790**
progl	71 646	2,039	3,278	3,243	2,135	2,134	**2,027**
progp	49 379	**2,010**	3,227	3,276	2,130	2,129	2,017
trans	93 695	1,835	2,942	2,977	1,991	1,960	**1,810**
průměr		2,700	4,005	4,294	2,814	2,769	**2,614**

Tabulka 4.4: Výsledky testování na korpusu Calgary(Huffmanovo kódování)



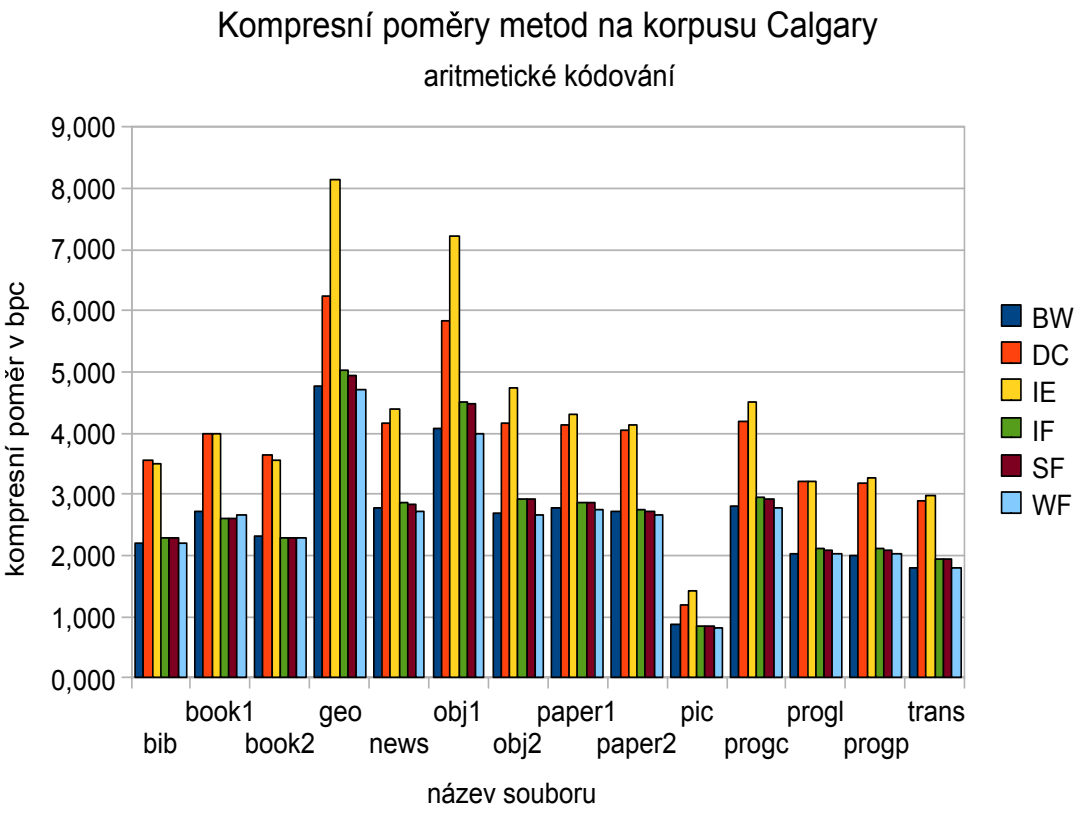
Obrázek 4.1: Výsledky testování na korpusu Calgary(Huffmanovo kódování)

V tabulce jsou tučně vyznačeny nejlepší výsledky pro daný soubor z Calgary korpusu. Nejlepší kompresní poměr byl dosažen s váženým frekvenčním počtem. Který měl téměř u všech souborů nejlepší kompresní poměr, jen s vyjímkou největších textových souborů, u kterých byl lepší řazený frekvenční počet. Celkově druhý nejlepší kompresní poměr byl naměřen s metodou přesuň na začátek. Tato metoda dosáhla nejlepšího výsledku u jednoho z nejmenších textových souboru *progp*.

V následující tabulce 4.5 jsou hodnoty z testování též Calgary korpusu. Jediným rozdílem oproti předchozí tabulce 4.4 je změna entropického kodéru na aritmetické kódování. Data z tabulky byly též znázorněny v grafu 4.2.

Soubor	Velikost	MTF	DC	IE	IF	SIF	WFC
bib	111 261	2,208	3,546	3,486	2,285	2,279	**2,192**
book1	768 771	2,713	3,986	4,003	2,599	**2,592**	2,650
book2	610 856	2,321	3,643	3,565	2,286	**2,278**	2,282
geo	102 400	4,775	6,243	8,146	5,036	4,943	**4,708**
news	377 109	2,766	4,176	4,391	2,856	2,847	**2,716**
obj1	21 504	4,072	5,822	7,229	4,496	4,481	**3,993**
obj2	246 814	2,689	4,149	4,731	2,932	2,918	**2,654**
paper1	53 161	2,773	4,119	4,313	2,876	2,861	**2,753**
paper2	82 199	2,707	4,055	4,142	2,748	2,727	**2,675**
pic	513 216	0,872	1,188	1,412	0,858	0,847	**0,830**
progc	39 611	2,814	4,199	4,493	2,945	2,931	**2,788**
progl	71 646	2,033	3,308	3,217	2,102	2,091	**2,023**
progp	49 379	**2,014**	3,172	3,266	2,108	2,099	2,018
trans	93 695	**1,795**	2,907	2,973	1,939	1,934	1,797
průměr		2,611	3,887	4,240	2,719	2,702	**2,577**

Tabulka 4.5: Výsledky testování na korpusu Calgary(aritmetické kódování)



Obrázek 4.2: Výsledky testování na korpusu Calgary(aritmetické kódování)

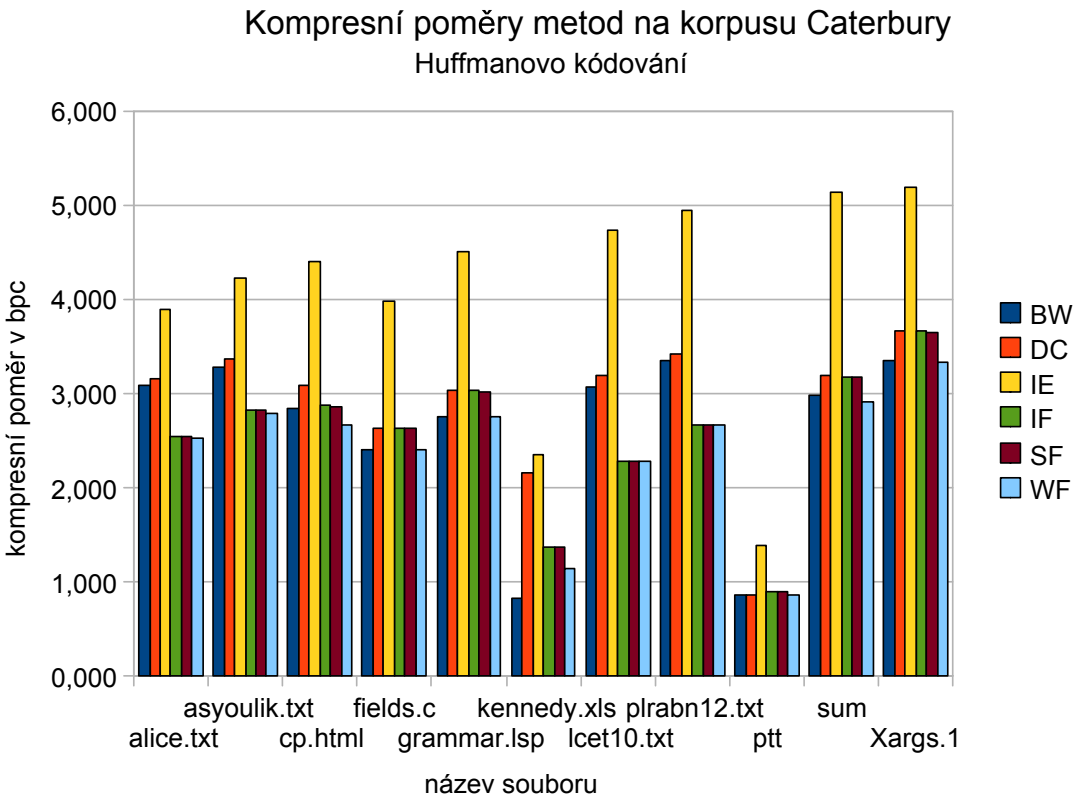
Z tabulky vidíme, že opět nejlépe dopadl vážený frekvenční počet následovaný přesuň na začátek. Metoda přesuň na začátek si připsala o jeden nejlepší kompresní poměr více oproti předchozímu měření a to u souboru *trans*. Celkově průměrný kompresní poměr byl snížen u všech testovaných metod a to díky použití aritmetického kódování.

4.2.2 Caterbury korpus

Druhým testovaným korpusem je Caterbury. Blíže byl popsán v sekci 4.1.2. Nejdříve byly zvolené metody testovány s Huffmanovým kódováním. Hodnoty kompresního poměru z testování jsou uvedeny v následující tabulce 4.6 a grafu 4.3.

Soubor	Velikost	MTF	DC	IE	IF	SIF	WFC
alice.txt	152 089	3,094	3,161	3,906	2,544	2,543	**2,539**
asyoulik.txt	125 179	3,285	3,375	4,224	2,825	2,825	**2,797**
cp.html	24 603	2,849	3,085	4,405	2,885	2,865	**2,663**
fields.c	11 150	**2,405**	2,639	3,980	2,639	2,637	2,414
grammar.lsp	3 721	**2,761**	3,031	4,517	3,031	3,014	2,765
kennedy.xls	1 029 744	**0,826**	2,163	2,351	1,380	1,380	1,155
lcet10.txt	426 754	3,083	3,202	4,733	2,286	2,286	**2,284**
plrabn12.txt	481 861	3,363	3,434	4,950	2,671	2,670	**2,670**
ptt5	513 216	0,863	0,872	1,392	0,897	0,896	**0,861**
sum	38 240	2,992	3,192	5,151	3,177	3,177	**2,916**
xargs.1	4 227	3,361	3,664	5,195	3,664	3,649	**3,333**
průměr		2,625	2,892	4,073	2,545	2,540	**2,400**

Tabulka 4.6: Výsledky testování na korpusu Caterbury(Huffmanovo kódování)



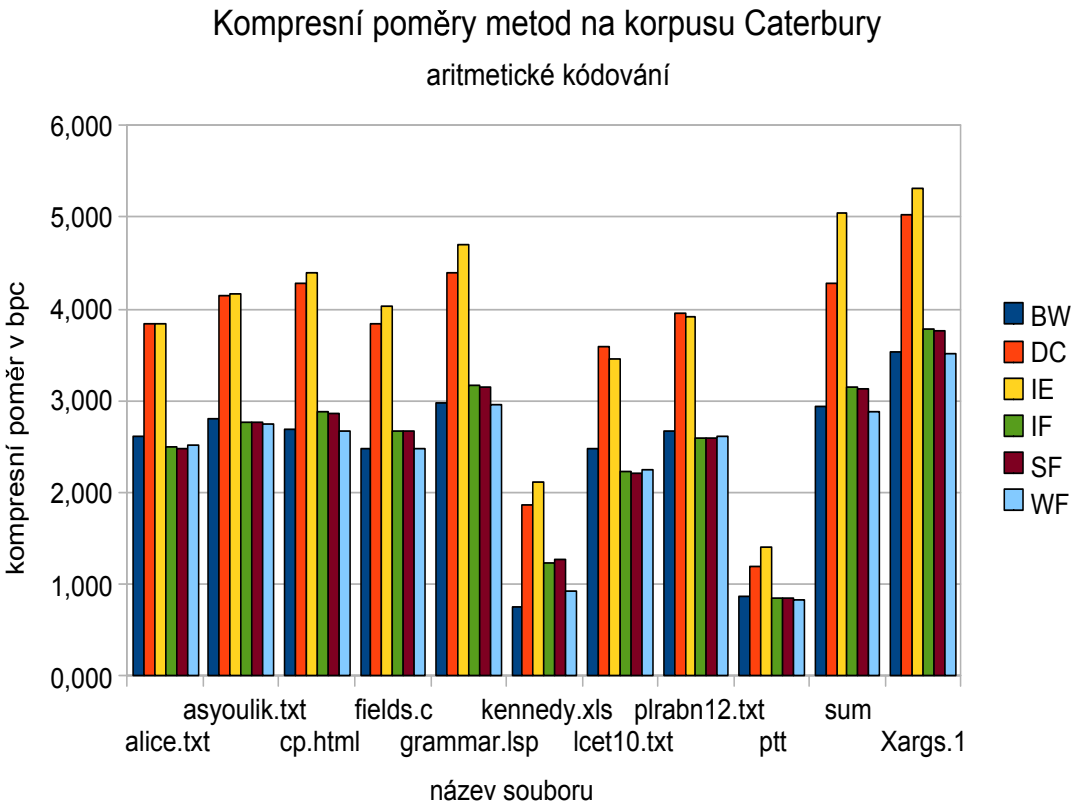
Obrázek 4.3: Výsledky testování na korpusu Caterbury(Huffmanovo kódování)

Z tabulky je vidět, že opět nejvýkonnější metodou byl vážený frekvenční počet. Dosáhl téměř u všech souborů nejlepší výsledky. Ve třech případech byla lepší metoda přesuň na začátek, největší rozdíl byl u binárního souboru *kennedy.xls*. Dále byla lepší u dvou menších textových souborů *fields.c* a *grammar.lsp*. I když metoda přesuň na začátek dosáhla tří nejlepších výsledků, tak v průměrném hodnocení byla horší než inverzní frekvence a řazená inverzní frekvence. Toto bylo způsobeno horšími výsledky u větších textových souborů. Na druhém a třetím místě tedy skončily metody řazená inverzní frekvence a inverzní frekvence, které dosahovali téměř totožných výsledků. Při porovnání s váženým frekvenčním počtem měli minimální rozdíly.

Druhé testování Caterbury korpusu bylo provedeno s aritmetickým kodérem. Stejně jako předchozí testování bylo i toto provedeno se stejnými metodami nad stejnými daty. Naměřená data byla zapsána do tabulky 4.7 a grafu 4.4.

Soubor	Velikost	MTF	DC	IE	IF	SIF	WFC
alice.txt	152 089	2,609	3,841	3,847	2,497	**2,482**	2,519
asyoulik.txt	125 179	2,801	4,141	4,168	2,771	2,758	**2,756**
cp.html	24 603	2,687	4,279	3,393	2,872	2,865	**2,668**
fields.c	11 150	2,475	3,834	4,037	2,668	2,667	**2,475**
grammar.lsp	3 721	2,971	4,399	4,696	3,171	3,154	**2,968**
kennedy.xls	1 029 744	**0,755**	1,868	2,120	1,234	1,275	0,930
lcet10.txt	426 754	2,472	3,581	3,452	2,222	**2,214**	2,244
plrabn12.txt	481 861	2,662	3,954	3,918	2,595	**2,586**	2,606
ptt5	513 216	0,872	1,188	1,414	0,858	0,847	**0,830**
sum	38 240	2,935	4,276	5,047	3,148	3,128	**2,882**
xargs.1	4 227	3,535	5,023	5,320	3,781	3,766	**3,511**
průměr		2,611	3,887	3,856	2,529	2,522	**2,399**

Tabulka 4.7: Výsledky testování na korpusu Caterbury(aritmetické kódování)



Obrázek 4.4: Výsledky testování na korpusu Caterbury(aritmetické kódování)

Opět díky aritmetickému kodéru došlo k mírnému zlepšení celkového průměrného kompresního poměru jednotlivých metod. Pořadí mezi průměrnými výsledky kompresních poměrů zůstalo na předních pozicích bezezměny. Vážený frekvenční počet dosáhl u dvou třetin souborů nejlepšího kompresního poměru. U zbylých souborů se jejich kompresní poměr výrazně přibližoval k těm nejlepším, které byli naměřeny. Metoda řazené inverzní frekvence dosáhla nejlepších výsledků u největších textových souborů. Přesuň na začátek si udržela nejlepší kompresní poměr u největšího binárního souboru.

4.2.3 Porovnání podle časové náročnosti

Dalšímu testování byly podrobeny jednotlivé kompresní metody a to jejich časové náročnosti. Nebyla měřena časová náročnost samotné metody, ale celého algoritmu s danou metodou. Předpokládanou nejrychlejší metodou je přesuň na začátek. Tato metoda oproti ostatním pouze přesouvá indexy a nevypočítává žádné hodnoty jako ostatní metody. Měření proběhlo pětkrát pro každou metodu a naměřené hodnoty byly zprůměrovány. Výsledky měření jednotlivých metod můžete vidět v tabulkách 4.8, 4.9. Hodnoty z tabulek byly vyvedeny do grafů 4.5, 4.6. Oběma grafům byla upravena časová osa, aby bylo možné vidět i menší rozdíly mezi jednotlivými metodami. Naměřené hodnoty jsou ve formátu mm:ss,00. Kurzivou byly zvýrazněny nejhorší výsledky.

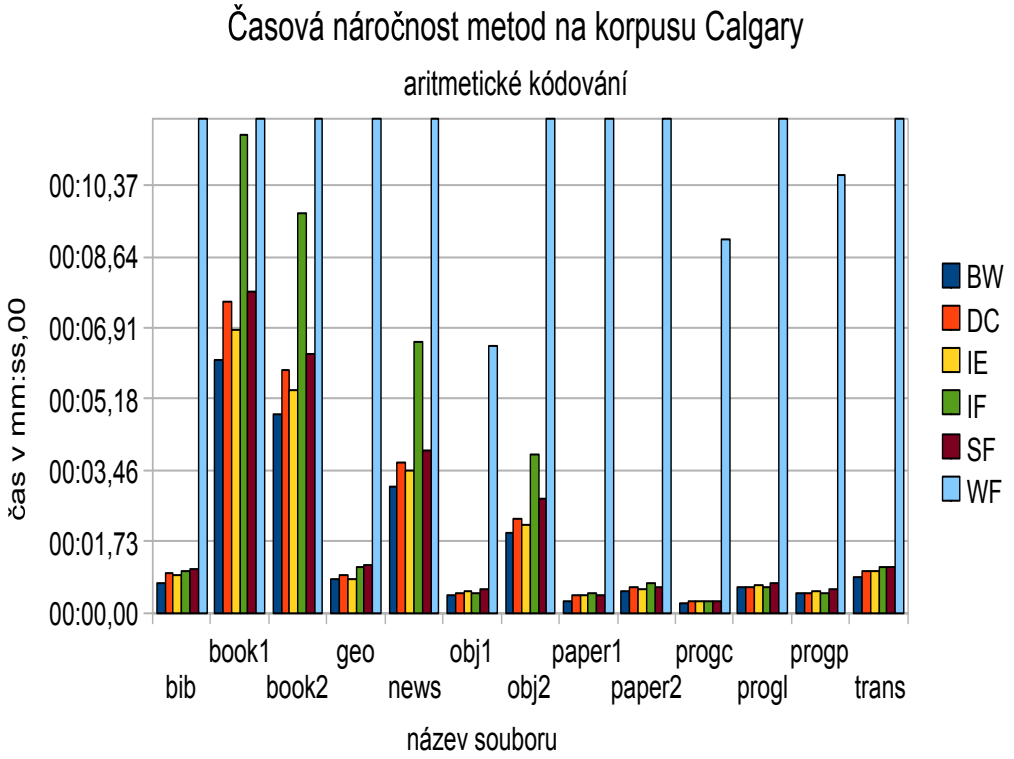
Soubor	Velikost	MTF	DC	IE	IF	SIF	WFC
bib	111 261	**00:00,71**	00:00,96	00:00,90	00:01,00	00:01,06	*00:23,60*
book1	768 771	**00:06,11**	00:07,55	00:06,88	00:11,59	00:07,81	*02:46,85*
book2	610 856	**00:04.80**	00:05,87	00:05,39	00:09,68	00:06,29	*02:10,92*
geo	102 400	00:00,82	00:00,89	**00:00,80**	00:01,08	00:01,13	*00:29,64*
news	377 109	**00:03,05**	00:03,63	00:03,44	00:06,57	00:03,94	*01:24,72*
obj1	21 504	**00:00,43**	00:00,45	00:00,50	00:00,49	00:00,54	*00:06,48*
obj2	246 814	**00:01,92**	00:02,26	00:02,15	00:03,85	00:02,78	*01:07,31*
paper1	53 161	**00:00,28**	00:00,43	00:00,43	00:00,47	00:00,42	*00:12,11*
paper2	82 199	**00:00,52**	00:00,63	00:00,56	00:00,70	00:00,61	*00:17,88*
pic	513 216	**09:05,63**	09:08,19	09:05,96	09:06,01	09:06,51	*10:51,55*
progc	39 611	**00:00,21**	00:00,27	00:00,26	00:00,29	00:00,25	*00:09,08*
progl	71 646	**00:00,59**	00:00,61	00:00,65	00:00,62	00:00,72	*00:15,59*
progp	49 379	**00:00,46**	00:00,49	00:00,51	00:00,49	00:00,55	*00:10,61*
trans	93 695	**00:00,88**	00:01,00	00:00,98	00:01,09	00:01,09	*00:21,45*

Tabulka 4.8: Měření časové naročnosti metod na Calgary korpusu (aritmetické kódování)

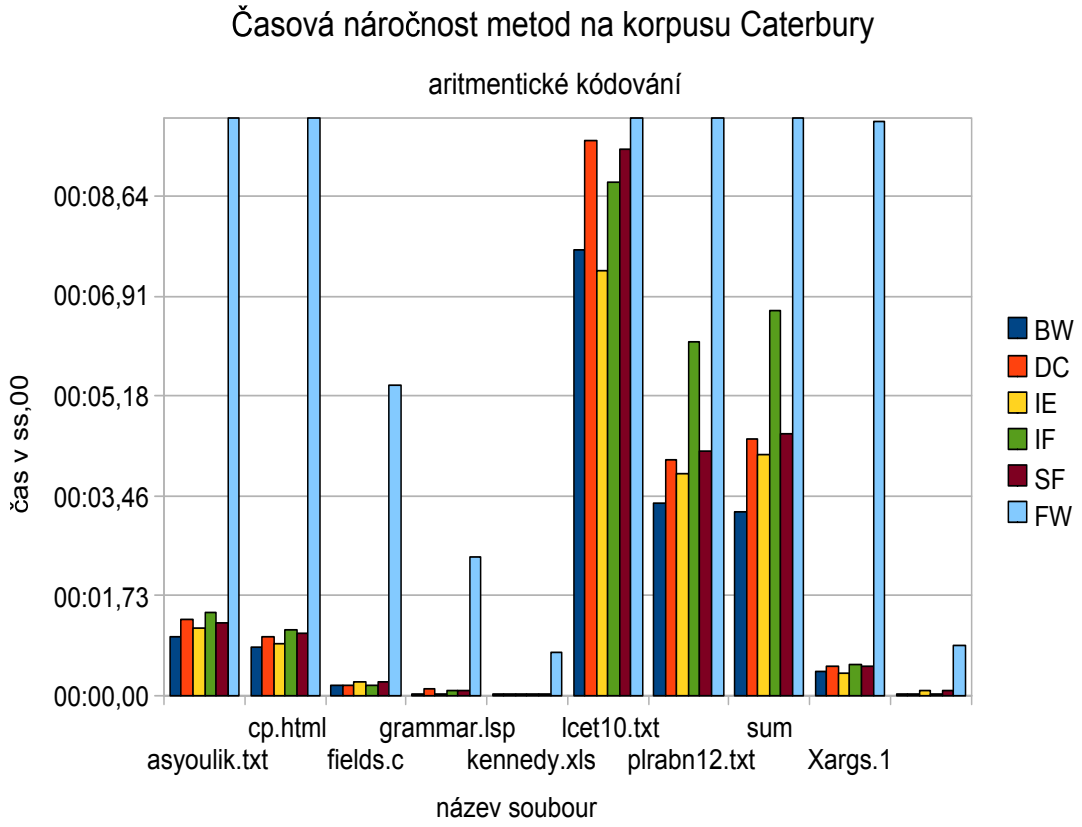
[h]

Soubor	Velikost	MTF	DC	IE	IF	SIF	WFC
alice.txt	152 089	**00:01,01**	00:01,30	00:01,16	00:01,43	00:01,26	*00:33,00*
asyoulik.txt	125 179	**00:00,82**	00:01,00	00:00,88	00:01,13	00:01,06	*00:26,73*
cp.html	24 603	00:00,18	**00:00,17**	00:00,23	00:00,17	00:00,23	*00:05,37*
fields.c	11 150	**00:00,02**	00:00,12	00:00,03	00:00,09	00:00,09	*00:02,38*
grammar.lsp	3 721	**00:00,01**	00:00,01	00:00,01	00:00,01	00:00,01	*00:00,75*
kennedy.xls	1 029 744	00:07,70	00:09,59	**00:07,35**	00:08,87	00:09,45	*03:29:28*
lcet10.txt	426 754	**00:03,32**	00:04,08	00:03,85	00:06,12	00:04,22	*01:29,61*
plrabn12.txt	481 861	**00:03,19**	00:04,43	00:04,17	00:06,66	00:04,54	*01:44,70*
ptt5	513 216	**09:05,75**	09:08,18	09:05,91	09:06,22	09:06,62	*10:45,55*
sum	38 240	00:00,41	00:00,51	**00:00,38**	00:00,52	00:00,49	*00:09,93*
xargs.1	4 227	**00:00,01**	00:00,01	00:00,07	00:00,01	00:00,07	*00:00,87*

Tabulka 4.9: Měření časové naročnosti metod na Caterbury(aritmetické kódování)



Obrázek 4.5: Měření časové naročnosti metod na Calgary korpusu (aritmetické kódování)



Obrázek 4.6: Měření časové naročnosti metod na Caterbury(aritmetické kódování)

Z obou tabulek je vidět, že předpoklad o časové náročnosti metod byl správný. Nejrychlejší metodou je přesuň na začátek. Nejhůře dopadl vážený frekvenční počet. Jeho časová náročnost převyšuje několikanásobně náročnosti ostatních metod. Toto navýšení je způsobeno porovnáváním jednotlivých prvků v plovoucím okně a vypočítáváním váhové hodnoty pro každý symbol vstupní abecedy v každém kroku. Tento obdobný problém je i v případě souboru *ppt5*, kde při porovnávání řetězců v Burrowsově-Wheelerově transformaci dochází k výraznému navýšení časové náročnosti.

4.2.4 Zhodnocení kompresních metod

Nyní si zhodnotíme celkově všechny testované metody. Protože by ideální metoda měla mít dobrý kompresní poměr, ale i zároveň by měla být rychlá, tak se musí oba parametry zohlednit při ohodnocení jednotlivých kompresních metod. Data, která budou brána pro porovnávání, budou z měření kompresního poměru s aritmetickým kodérem viz. tabulka 4.5. Data pro časovou náročnost budou brána z tabulky 4.8. Porovnání dat proběhlo podle následujících kroků. Vypočítal se medián kompresního poměru pro jednotlivé soubory. Tímto mediánem byl podělen původní kompresní poměr. Následně pro každou metodu bylo provedeno zprůměrování takto získaného koeficientu komprese. Výpočtené hodnoty koeficientu komprese dat vidíme v následující tabulce 4.10.

Obdobným způsobem, jako jsme vypočítali koeficient komprese dat, vypočítáme i koefi-

Soubor	Medián	MTF	DC	IE	IF	SIF	WFC
bib	2,282	0,968	1,554	1,528	1,001	0,999	0,961
book1	2,682	1,012	1,486	1,493	0,969	0,967	0,988
book2	2,303	1,008	1,582	1,548	0,992	0,989	0,991
geo	4,990	0,957	1,251	1,633	1,009	0,991	0,994
news	2,851	0,970	1,465	1,540	1,002	0,998	0,953
obj1	4,489	0,907	1,297	1,611	1,002	0,998	0,890
obj2	2,925	0,919	1,419	1,617	1,002	0,998	0,907
paper1	2,869	0,967	1,436	1,503	1,003	0,997	0,960
paper2	2,737	0,989	1,481	1,513	1,004	0,996	0,977
pic	0,865	1,008	1,373	1,632	0,992	0,979	0,959
progc	2,938	0,958	1,429	1,529	1,002	0,998	0,949
progl	2,097	0,970	1,530	1,534	1,003	0,997	0,965
progp	2,103	0,957	1,508	1,553	1,002	0,998	0,959
trans	1,936	0,927	1,501	1,535	1,001	0,999	0,928
koeficient		0,965	1,451	1,555	0,999	0,993	0,952

Tabulka 4.10: Výpočet koeficientu komprese dat pro porovnání kompresních metod

cient časové náročnosti. Vypočítáme si medián pro jednotlivé soubory. Takto vypočítaným mediánem podělíme hodnotu naměřeného času trvání komprese dat dané metody. Pro jednotlivé metody vypočítáme průměr, který nám určuje koeficient časové náročnosti.

Soubor	Medián	MTF	DC	IE	IF	SIF	WFC
bib	00:00,98	0,724	0,980	0,918	1,020	1,082	24,082
book1	00:07,68	0,796	0,983	0,896	1,509	1,017	21,725
book2	00:06,08	0,789	0,965	0,887	1,592	1,035	21,533
geo	00:00,99	0,832	0,904	0,812	1,096	1,147	30,091
news	00:03,79	0,806	0,959	0,909	1,736	1,041	22,383
obj1	00:00,50	0,869	0,909	1,010	0,990	1,091	13,091
obj2	00:02,52	0,762	0,897	0,853	1,528	1,103	26,710
paper1	00:00,43	0,651	1,000	1,000	1,093	0,977	28,163
paper2	00:00,62	0,839	1,016	0,903	1,129	0,977	28,839
pic	09:06,26	0,999	1,016	0,999	1,000	1,000	1,193
progc	00:00,27	0,792	1,019	0,981	1,094	0,943	34,264
progl	00:00,64	0,929	0,961	1,024	0,976	1,134	24,551
progp	00:00,50	0,920	0,980	1,020	0,980	1,100	21,220
trans	00:01,05	0,842	0,957	0,938	1,043	1,043	20,526
koeficient		0,825	0,967	0,939	1,199	1,050	22,741

Tabulka 4.11: Výpočet koeficientu časové náročnosti pro porovnání kompresních metod

Koeficient účinnosti dané metody se pak spočítá jako průměr koeficientu komprese dat z tabulky 4.10 a koeficientu časové náročnosti z tabulky 4.11. Výsledné hodnoty byly zhrnuty v tabulce 4.12, které udávají účinnost jednotlivých kompresních metod. Hodnota účinnosti rovna jedné určuje hranici. Výsledky pod touto hranicí dosáhly lepších výsledků než průměrná metoda.

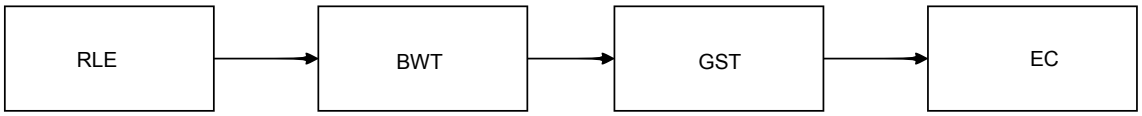
Metoda	MTF	DC	IE	IF	SIF	WFC
účinnost	0,895	1,209	1,247	1,099	1,021	11,846

Tabulka 4.12: Účinnost kompresních metod

Z tabulky 4.12 vyplývá, že nejúčinnější metodou je přesuň na začátek, která je následována řazeným inkrementačním počtem. Metoda přesuň na začátek je jednoduchou metodou, která nemá velké paměťové ani výpočetní nároky. Nedosahovala nejlepších výsledků v kompresním poměru, ale tuto ztrátu vynahradila svojí rychlostí, díky své jednoduchosti. Pomyslné druhé místo obsadila metoda řazené inverzní frekvence, která je modifikací inverzní frekvence. A potvrdila, že tato úprava původního algoritmu, dává lepší výsledky. Třetí skončila původní neupravená metoda inverzní frekvence. S větším odstupem pak skončilo distanční kódovaní. Následované intervalovým kódováním. Na posledním místě se umístil vážený frekvenční počet, díky jeho časové náročnosti.

4.3 Změna pořadí metod

Tato část se zaměřuje na změnu pořadí metod v Burrowsově-Wheelerově kompresní algoritmu. Jedná se o přesun metody kódování délkou sledů před Burrowsovu-Wheelerovu transformaci. Tato modifikace se využívá ke zvýšení rychlosti celého kompresního algoritmu. Tento posun metod je znázorněn na obrázku 4.8.



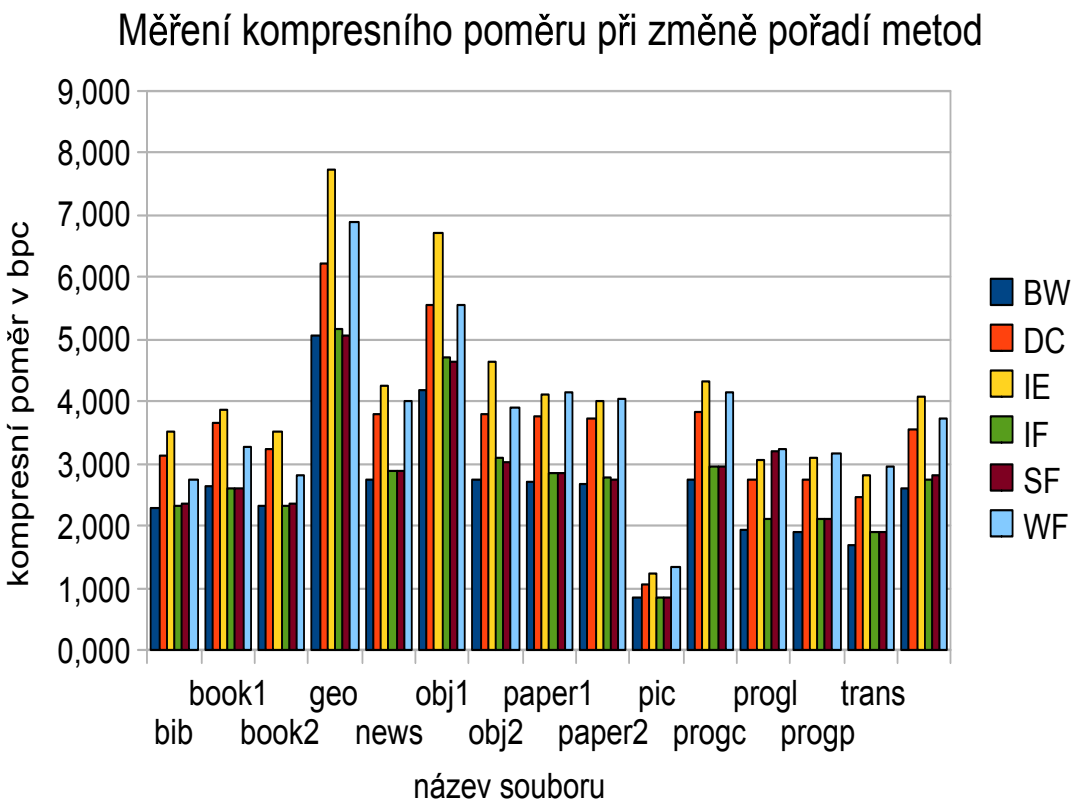
Obrázek 4.7: Upravený Burrowsův-Wheelerův kompresní algoritmus

4.3.1 Výsledky měření

Měření probíhalo na korpusu Calgary. Testy proběhly se všemi metodami transformace globální struktury. Nejlepší výsledky byly vyznačeny tučne v tabulkách. Naměřené hodnoty kompresního poměru jsou v tabulce 4.13 a grafu 4.8. Hodnoty naměřené časové náročnosti jsou v tabulce 4.14 a grafu 4.9.

Soubor	Velikost	MTF	DC	IE	IF	SIF	WFC
bib	111 261	**2,300**	3,139	3,497	2,333	2,337	2,749
book1	768 771	2,644	3,661	3,873	**2,591**	2,602	3,262
book2	610 856	2,644	3,661	3,873	**2,591**	2,602	3,262
geo	102 400	5,044	6,210	7,725	5,173	**5,045**	6,875
news	377 109	**2,750**	3,777	4,243	2,868	2,877	4,020
obj1	21 504	**4,169**	5,565	6,719	4,697	4,638	5,545
obj2	246 814	**2,751**	3,783	4,630	3,073	3,006	3,896
paper1	53 161	**2,697**	3,752	4,122	2,852	2,840	4,142
paper2	82 199	**2,683**	3,713	4,018	2,759	2,750	4,034
pic	513 216	0,846	1,062	1,237	**0,832**	0,833	1,316
progc	39 611	**2,749**	3,831	4,316	2,953	2,932	4,147
progl	71 646	**1,939**	2,751	3,071	2,109	3,210	3,228
progp	49 379	**1,899**	2,738	3,106	2,113	2,099	3,159
trans	93 695	**1,671**	2,441	2,815	1,893	1,880	2,958
průměr		**2,603**	3,547	4,062	2,755	2,814	3,725

Tabulka 4.13: Výsledky testování změny pořadí metod (kompresní poměr)



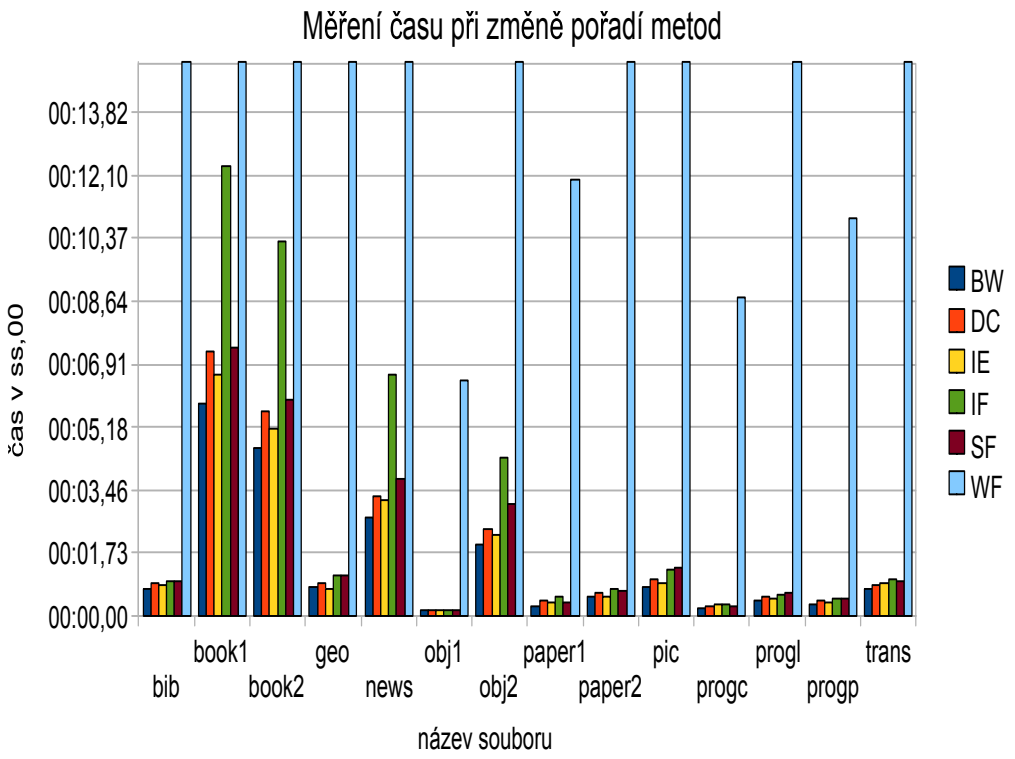
Obrázek 4.8: Výsledky testování změny pořadí metod (kompresní poměr)

Z tabulky 4.13 s naměřenými daty kompresního poměru je vidět, že jednoznačně nejlepší metodou je přesuň na začátek. Druhá skončila inverzní frekvence následovaná řazenou inverzní frekvencí. Při srovnání kompresních poměrů s nemodifikovanou verzí Burrowsova-Wheelerova kompresního algoritmu zjistíme, že metody přesuň na začátek, distanční a intervalové kódování dosáhlo lepších výsledků. Bohužel zbývající tři metody dosáhly horšího kompresního poměru a nejhuře dopadl vážený frekvenční počet.

Soubor	Velikost	MTF	DC	IE	IF	SIF	WFC
bib	111 261	**00:00,74**	00:00,88	00:00,82	00:00,93	00:00,95	00:27,14
book1	768 771	**00:05,82**	00:07,25	00:06,62	00:12,35	00:07,36	03:05,39
book2	610 856	**00:04,59**	00:05,62	00:05,11	00:10,29	00:05,94	02:10,51
geo	102 400	**00:00,80**	00:00,89	00:00,75	00:01,12	00:01,11	00:29,43
news	377 109	**00:02,70**	00:03,30	00:03,17	00:06,64	00:03,74	01:27,18
obj1	21 504	00:00,13	00:00,14	00:00,13	**00:00,12**	00:00,12	00:06,48
obj2	246 814	**00:01,96**	00:02,35	00:02,23	00:04,36	00:03,07	01:05,52
paper1	53 161	**00:00,27**	00:00,39	00:00,35	00:00,53	00:00,37	00:11,96
paper2	82 199	**00:00,50**	00:00,60	00:00,54	00:00,71	00:00,69	00:17,95
pic	513 216	**00:00,79**	00:00,97	00:00,90	00:01,27	00:01,30	10:48,54
progc	39 611	**00:00,20**	00:00,25	00:00,28	00:00,31	00:00,24	00:08,74
progl	71 646	**00:00,39**	00:00,54	00:00,48	00:00,56	00:00,64	00:15,35
progp	49 379	**00:00,28**	00:00,41	00:00,36	00:00,44	00:00,47	00:10,92
trans	93 695	**00:00,73**	00:00,81	00:00,90	00:00,97	00:00,96	00:20,21

Tabulka 4.14: Výsledky testování změny pořadí metod (časová náročnost)

V grafu 4.9 muselo být upraveno měřítko na ose s časem, aby bylo možné zobrazit rozdíly mezi menšími hodnotami u jednotlivých metod. Hodnoty naměřené u časové náročnosti jsou téměř stejné jako u nemodifikované verze až na jednu vyjímku. Tato vyjímka se týká souboru *pic*, kde došlo k výraznému zlepšení. Kde v nemodifikované variantě doba zpracování trvala několik minut tak zde byl tento soubor u většiny metod zpracován do jedné vteřiny.



Obrázek 4.9: Výsledky testování změny pořadí metod (časová náročnost)

4.4 Porovnání entropických kodérů

V této části porovnáme tři implementované entropické kodéry, které byly popsány výše. Jedná se o Huffmanovo kódování, které by mělo být v tomto porovnání lepší než Riceovo-Golombovo kódování, ale horší oproti aritmetickému kódování. Aritmetické kódování by mělo dosahovat nejlepších výsledků. Riceovo-Golombovo kódování je nejsilnější v kódování souborů s geometrickým rozložením symbolů. Proto v porovnání se zbylými entropickými kodéry nedopadne nejlépe na testovaných datech. Protože z větší části testované soubory nesplňují podmínku nejvhodnějšího použití tj. geometrické rozložení dat. Testování bylo provedeno na Calgary korpusu. Metoda zvolená pro transformaci globální struktury byla vážený frekvenční počet.

Naměřená data v tabulce 4.15 potvrdila, že aritmetické kódování dosahuje nejlepších výsledků z testovaných entropických kodérů. Podle předpokladů druhé skončilo Huffmanovo kódování. Nejhůře skončilo Riceovo-Golombovo kódování.

4.5 Porovnání nastavení Burrowsovy-Wheelerovy transformace

Tato část se zaměřuje na testování samotné Burrowsovy-Wheelerovy transformace. Není moc atributů, které mohou ovlivnit výkonnost této transformace. Mezi dvě varianty úprav Burrowsovy-Wheelerovy transformace můžeme zahrnout řadící algoritmus, na kterém vý-

Soubor	Huffmanovo kódování	Aritmeticke kódování	Riceovo-Golombovo kódování
bib	2,217	2,192	3,312
book1	2,728	2,650	4,114
book2	2,360	2,282	3,572
geo	4,839	4,708	6,875
news	2,781	2,716	4,020
obj1	3,981	3,993	5,545
obj2	2,715	2,654	3,896
paper1	2,776	2,753	4,142
paper2	2,700	2,675	4,034
pic	0,861	0,830	1,316
progc	2,790	2,788	4,145
progl	2,027	2,023	3,228
progp	1,810	2,018	3,159
trans	2,614	1,797	2,958
průměr	2,700	2,611	3,880

Tabulka 4.15: Výsledky testování entropických kodérů na korpusu Calgary

razně závisí časová náročnost transformace. Při Burrowsově-Wheelerově transformaci velkých souborů není vhodné provádět řazení celého souboru najednou. Mnohem vhodnější je zpracovávat větší soubory po blocích.

Tato část se právě zaměřuje na testování druhé varianty a to velikosti zpracováváných dat. Implementace této úpravy je jednoduchá. Ukážeme si ji na následujícím pseudokódu.

```
buffer[Maximalni_velikost_bufferu];

while( NeniKonecSouboru() )
{
  NactiZnakAUloz(buffer);
  PocetNactenychZnaku++;

 if(PocetNactenychZnaku == Maximální_velikost_bufferu)
 {
    BurrowsovaWheelerovaTransformace(buffer);

    VynulujBuffer(buffer);
    PocetNactenychZnaku = 0;
 }
}

BurrowsWheelerovaTransformace(buffer);
```

Pseudokód 4.1: Kódování pomocí Burrowsovy-Wheelerovy transformace

V pseudokódu funkce *BurrowsovaWheelerovaTransformace* představuje stejnou funkci, jako je popsána v kapitole 3.1. Obdobný kód se použije též i při dekompresi jen s tím rozdílem, že místo funkce, která komprimuje data, by byla fuknce dekomprimující data. Záleží na vhodně zvolené velikosti maximálně zpracovaných dat. Protože pokud zvolíme malou hodnotu bude

algoritmus rychlý, ale kompresní výkon bude horší. V opačném případě, pokud zvolíme příliš velkou hodnotu, získáme lepší kompresní výkon, ale časová náročnost se podstatně zvýší. Proto je potřeba najít kompromis mezi časovou a kompresní výkonností.

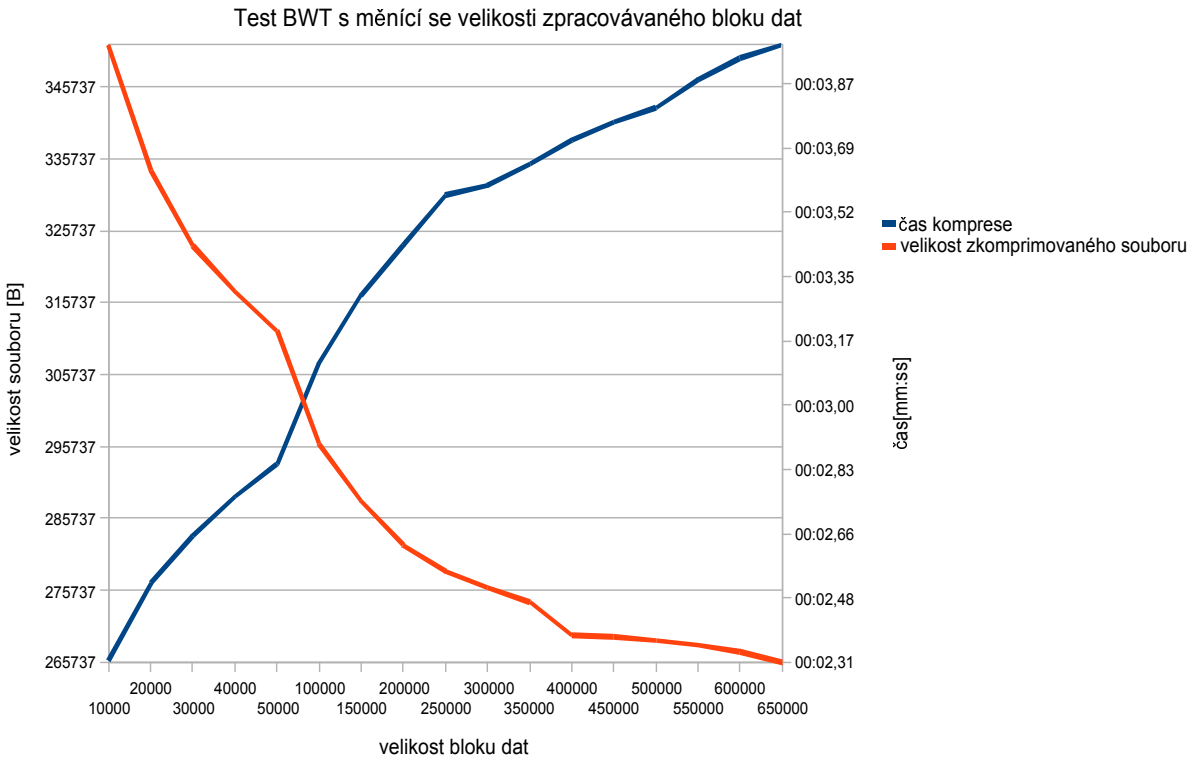
4.5.1 Testování velikosti zpracovaných dat

Testování velikosti zpracovaných dat probíhalo na posloupnosti transformací Burrowsova-Wheelerova, přesuň na začátek, kódování délkou sledů a aritmetické kódování. Souborem, na kterém bude prováděno testování, byl zvolen *book1* z korpusu Calgary. Velikost zpracovávaných dat začínala na hodnotě 10 000 znaků a končila na 650 000. V průběhu byl měřen čas, za který byla provedena komprese vstupních dat a kompresní poměr v bpc. Čas byl měřen pomocí příkazu *time*. Z možností naměřených hodnot byla vybrána ta, která byla naměřena pro *user time*. Bylo provedeno vždy pět testů a naměřené časové hodnoty byly zprůměrovány. Hodnoty, které byly naměřeny, jsou v tabulce 4.16.

Velikost bloku	Doba zpracování	Velikost komprimovaného souboru	Kompresní poměr
10 000	2,314 s	351 584	3,66
20 000	2,523 s	334 050	3,48
30 000	2,650 s	323 667	3,37
40 000	2,756 s	317 275	3,30
50 000	2,843 s	311 806	3,24
100 000	3,114 s	296 185	3,08
150 000	3,297 s	288 119	3,00
200 000	3,433 s	282 014	2,93
250 000	3,566 s	278 396	2,90
300 000	3,593 s	276 110	2,87
350 000	3,650 s	274 112	2,85
400 000	3,713 s	269 516	2,80
450 000	3,763 s	269 252	2,80
500 000	3,801 s	268 805	2,80
550 000	3,875 s	268 169	2,79
600 000	3,935 s	267 204	2,78
650 000	3,973 s	265 737	2,77

Tabulka 4.16: Naměřené hodnoty při testování Burrowsovy-Wheelerovy transformace a velikosti zpracovaných dat

Z důvodu zmenšujícího se kompresního poměru mezi jednotlivými velikostmi zpracovaných dat, byla tabulka rozšířena o sloupec *Velikost komprimovaného souboru*. Díky němu jsou patrná i menší zlepšení kompresního poměru. Naměřené hodnoty byly pro lepší představení, jak nárůstu časové náročnosti, tak klesání velikosti zkomprimovaného souboru vyvedeny do grafu 4.10.



Obrázek 4.10: Naměřené hodnoty při testování Burrowsovy-Wheelerovy transformaci a velikosti zpracovaných dat

Z grafu i tabulky vidíme, že naměřená doba zpracování roste s velikostí zpracovávaných dat. Největší podíl na časovém růstu má předpříprava a porovnávání dat v Burrowsově-Wheelerově metodě. Teď se budeme soustředit na kompresní poměr respektive velikost zkomprimovaného souboru. Vidíme, že rozdíl v kompresním poměru mezi nejmenším blokem dat a tím největším, je téměř 0,85 bpc. Kompresní poměr od velikosti bloku 10 000 až po 400 000 rychle klesá. Ale již okolo hodnoty 400 000 začíná tato hodnota stagnovat. Proto ideální hodnotou bloku, dle naměřených dat, je tedy velikost okolo 400 000-500 000, i když doba komprese vstupních dat vzrostla.

4.6 Porovnání s jinými kompresními programy

Nejdříve si popíšeme jednotlivé kompresní programy, které jsou aktuálně jedny z nejpoužívanějších. Programy pro kompresi jsem zvolil 7-zip a WinRAR. Tyto dva programy umožňují komprimovat data do nejpoužívanějších formátů. Oba programy jsou multiplatformní. Program WinRAR je shareware. Poprvé byl vydán v roce 1993. Program 7-zip je freeware pod licencí GNU LGPL. Z programu 7-zip budou použity následující kompresní formáty *7z*, *bzip2*, *gzip*, *zip*. Program WinRAR poslouží ke komprimaci do formátu *rar*.

4.6.1 Popis formátů

Zde budou detailněji popsány jednotlivé formáty, které budou využity k testování. Prvním formátem je *rar*. Ve verzi 3.0 je založena na predikci podle částečné shody (jeho variantě

PPMII) a Lempel-Ziv(LZSS). Metoda LZSS patří mezi nejznámější slovníkové metody. Druhým formátem je *7z*. U tohoto formátu lze v programu 7-zip nastavit několik komprimačních metod. Mezi nastavitelné metody patří slovníkové metody LZMA, LZMA2, PPMd a metoda BZip2. Metoda BZip2 je založena na Burrowsově-Wheelerově transformaci s přesuň na začátek a Huffmanovým kódováním. Předchůdce Bzip měl namísto Huffmanova kódování Aritmetické kódování. Následujícím formátem je *gzip*, který používá kompresní metodu Deflate. Tato kompresní metoda je založena na slovníkové metodě LZ77 a Huffmanově kódování. Posledním zbývajícím je *zip*, který využívá obdobně jako *7z* metody LZMA, PPMd a BZip2, ale i Deflate a Deflate64.

4.6.2 Výsledky

Pro testování Burrowsova-Wheelerova algoritmu byla zvolena za transformaci globální struktury metoda přesuň na začátek a entropický kodér aritmetické kódování. U formátu *7z* byla zvolena metoda LZMA2 a pro *zip* metoda LZMA. Ostatní algoritmy byly nechány v původním nastavení. Po otestování všech algoritmů byly naměřeny následující hodnoty kompresního poměru, které jsou zobrazeny v tabulce 4.17.

Soubor	Velikost	BWTA	7z	BZip2	gzip	zip	rar
dickens	10 192 446	2,486	2,222	**2,199**	2,893	2,222	2,453
mozilla	51 220 480	3,013	**2,111**	2,798	2,899	2,111	2,431
mr	9 970 564	2,053	2,212	**1,962**	2,800	2,209	2,593
nci	33 553 445	0,530	**0,472**	0,473	0,725	0,473	0,517
ooffice	6 152 192	3,954	3,158	**2,423**	3,915	3,158	3,015
osdb	10 085 684	2,368	2,278	**2,226**	2,879	2,278	2,602
reymont	6 627 202	1,726	1,622	**1,508**	2,062	1,622	1,889
samba	21 606 400	1,862	**1,430**	1,685	1,923	1,430	1,564
sao	7 251 944	5,746	**4,871**	5,454	5,788	4,875	6,145
webster	41 458 703	1,893	1,699	**1,668**	2,249	1,698	1,872
xml	5 345 280	0,776	0,730	**0,662**	0,999	0,730	0,754
xray	8 474 240	4,023	4,238	**3,828**	5,441	4,238	3,905
průměr		2,536	2,253	**2,240**	2,881	2,253	2,478

Tabulka 4.17: Výsledky testování na korpusu Caterbury(Aritmetické kódování)

Z výsledků v tabulce 4.17 vidíme, že nejlepších výsledků docílil *BZip2*. Těsně následovaný *7z* a *zip*. Téměř se srovnatelným výsledkem skončil Burrowsův-Wheelerův algoritmus za algoritmem *rar*. Poslení skončil *gzip*. Burrowsův-Wheelerův algoritmus dosahoval srovnatelných výsledků s nejlepšími algoritmy u obrázků *mr* a *xray*. Bohužel, nejhorší výsledky byly naměřeny u spustitelných binárních souborů *mozilla*, *oofice* a kalendáře *sao*. Burrowsův-Wheelerův algoritmus se tedy zařadil k horšímu průměru porovnávaných algoritmů.

Kapitola 5

Závěr

Tato diplomová práce byla zaměřena na popis a implementaci statistických kompresních metod. Přesněji na Burrowsův-Wheelerův algoritmus a s ním související modifikace. Byla vytvořena knihovna, ve které byly implementovány metody, které modifikují druhý stupeň (transformace globální struktury) Burrowsova-Wheelerova kompresního algoritmu popsaného v kapitole 3.5. Zároveň umožňuje volbu entropického kodéru. Na výběr je aritmetické, Huffmanovo nebo Riceovo-Golombovo kódování. Popis jednotlivých entropických kodérů je v kapitole 3.4.

Implementované metody transformace globální struktury a entropické kodéry byly testovány na korpusech Calgary a Caterbury. Testování metod transformace globální struktury probíhalo nejdříve na Calgary korpusu. Na tomto korpusu dosáhla nejlepšího kompresního poměru metoda váženého frekvenčního počtu. Tato metoda v rámci tohoto korpusu ukázala, že je univerzální metodou pro různé typy souborů. Pouze u největších textových souborů z toho korpusu dosáhla lepších výsledků metoda řazené inverzní frekvence. Obdobných výsledků bylo dosaženo i při testování Caterbury korpusu. Při zhodnocení výsledků jednotlivých metod se musí vzít v úvahu i jejich časová náročnost. Proto při celkovém zhodnocení metod byl zahrnut i tento faktor. Výsledky zhodnocení jednotlivých metod nalezneme v tabulce 4.12. I když nejlepších kompresních poměrů dosahovala metoda váženého frekvenčního počtu, tak po přidání časové náročnosti, jako dalšího hodnotícího faktoru, spadla až na poslední místo. Je to díky její výpočetní náročnosti, která je způsobena nutnými výpočty frekvenčních vah po každém načtení znaku. Proto nejlépe hodnocenou metodou byla přesuň na začátek, následovaná řazenou inverzní frekvencí.

Testování etropických kodérů probíhalo na Calgary korpusu. Testování metod trasnformace globální struktury probíhalo s dvěmi entropickými kodéry (aritmetické a Huffmanovo kódování). Zde bylo přidáno k testování i Riceovo-Golombovo kódování. Z naměřených hodnot, které byly uvedeny v tabulce 4.15, je patrné, že nejlepšího kompresního poměru dosáhlo aritmetické kódování, které bylo o 3% lepší než Huffmanovo kódování. Podle předpokladů nejhůře skončilo Riceovo-Golombovo kódování, které dosahuje ideálních výsledků, pokud vstupní text má geometrické rozložení symbolů.

Byla testována varianta se záměnou metod v Burrowsově-Wheelerově kompresním algoritmu. Při této konfiguraci došlo u tří metod k zlepšení kompresního poměru vůči nemodifikované variantě. Největší zhoršení měl vážený frekvenční počet. U časové náročnosti byly data obdobná jen u souboru *pic* došlo k zrychlení o několik řádů.

Při testování velikosti dat, které budou řazeny v Burrowsově-Wheelerově transformaci popsané v kapitole 4.5, byla provedena měření s velikostmi dat od 10 000 až 650 000 viz. tabulka 4.16. Ideální velikost dat pro řazení je mezi 400 000-500 000, kde kompresní poměr

začíná stagnovat.

Implementovaný Burrowsův-Wheelerův kompresní algoritmus byl porovnán s nejmpoužívanějšími kompresními algoritmy, které jsou popsány v kapitole 4.6. Testování proběhlo na korpusu Silecia. Při testování Burrowsův-Wheelerův kompresní algoritmus dosáhl téměř shodného výsledku, jako kompresní algoritmus *rar*, čímž se zařadil mezi horší průměr měřených algoritmů v porovnání kompresního poměru. Nejlepšího kompresního poměru dosáhl u obrázku magnetické rezonance.

Literatura

[1] Abel, J.: Improvements to the Burrows-Wheeler Compression Algorithm: After BWT Stages. 2003, university Duisburg-Essen.

[2] Abel, J.: *Incremental frequency count - a post BWT-stage for the Burrows-Wheeler compression algorithm*. Software: Practice and Experience 32(2), 2007, 247-265 s.

[3] Abel, J.: *Post BWT stages of the Burrows Wheeler compression algorithm*. Software: Practice and Experience 40(9), 2010, 751-777 s.

[4] Bodden, M., E.; Clasen; Kneis, J.: Arithmetic Coding revealed - a guided tour from theory to praxis. Technická zpráva, Aachen University, 2007-5, sable (2007).

[5] Burrows, M.; Wheeler, D. J.: A block-sorting lossless data compression algorithm. Technická zpráva, 1994.

[6] Deorowicz, S.: *Second step algorithms in the Burrows-Wheeler compression algorithm*. Software: Practice and Experience 32(2), 2002, 99-111 s.

[7] Drábek, V.: *Kódování a komprese dat*. Brno: Vysoké učení technické v Brně, 2008.

[8] Elias, P.: *Interval and Recency Rank Source Coding: Two On-Line Adaptive Variable-Length Scheme*. Proceedings of the IEEE Data Copression Conference, 1987, 194-203 s.

[9] Said, A.: Resilient parameterized tree codes for fast adaptive coding. Technická zpráva, HP Laboratories Palo Alto, 2004 [cit. 2013-04-20].
URL http://www.hpl.hp.com/techreports/2004/HPL-2004-102.pdf

[10] Večerka, A.: Komprese dat. online, 2008, Univerzita Palackého.
URL http://phoenix.inf.upol.cz/esf/ucebni/komprese.pdf

Příloha A

Obsah CD

- Technická zpráva ve formátu PDF
- Zdrojové soubory k technické zprávě ve formátu LaTeX
- Zdrojové soubory ke konzolové aplikaci
- Programová dokumentace ve formátu html vygenerováná programem Doxygen
- Soubory korpusu Calgary, Caterbury a Silecia